ANR IMPRO
Deliverable 5.1
Control tasks for Timed System; Robustness issues

Patricia Bouyer, Sébastien Faucou, Stefan Haar,
Aleksandra Jovanović, Didier Lime, Nicolas Markey,
Olivier H. Roux and Ocan Sankur

January 29, 2013

1 Introduction

The necessity to observe, monitor and control complex systems, combined with the difficulty of partial observation, are a pervasive subject of system theory, from continuous through discrete settings. We address here two supervisory tasks in the setting of *timed* discrete event systems:

- **Diagnosis**: Given a TDES S that is partially observable and some unobservable property ϕ (a *fault condition*) on S, use the partial observation of S's behaviour to determine whether ϕ is true or not;
- **Control**: Given S as above and a target property ψ for S, use the partial observation of S's behaviour *and* a set of allowed control actions on S to ensure S satisfies ψ.

Here, the observation can concern *states* or *events* of S, and control may be exerted by blocking one or more *controllable* transitions of S. Typically, ϕ represents the occurrence of particular unobservable *fault* events, and ψ that the system is in some set of allowed states or allowed behaviours (e.g. expressed in a suitable temporal logic).

Robustness. The synthesis tasks above may not be solvable for a given DES in the sense that

- observation may be too sparse or too ambiguous to discern faulty from non-faulty behaviour, and
- controllable actions may be too few, or too badly distributed , to allow for effective control in the above sense.

While these issues remain important when *timed* DES are considered, both tasks then also need to face several *robustness* problems:

- *Concurrent* behaviours may weaken the capacity of observation, and lead to a modification of the type of fault conditions that can be diagnosed (haar-tac10) and of properties that can be ensured via control ; the fine analysis of partial order semantics is crucial to capture this effect and make supervision robust.
- *Loss* or falsification of observations are notoriously hard to handle in supervision; systems and supervisors need to be robust w.r.t. such failures in both the event *and* the time domain.
- *Imprecisions* of behaviour, together with the impossibility to implement sharp time intervals or points in real-time equipments, pose problems at the very *simulation* of behaviour.

In the following, we look at some of these aspects in more detail, give a sketch of existing literature, and propose some research problems to address.

2 Diagnosis

2.1 The Problems of Diagnosis and Diagnosability

The literature on diagnosis in the above sense for *untimed* discrete event systems is vast; see [CL08] for a detailed and "canonical" introduction. Below follows a survey of approaches to extend to *timed* discrete event systems the techniques of diagnosis for DES, as well as the verification of the possibility of diagnosis, i.e. *diagnosability.* Fix an alphabet Σ and a subset $\Sigma_O \subseteq \Sigma$ of *observable* letters; this defines the set of *unobservable* letters (or events) as $\Sigma_U \triangleq \Sigma \backslash \Sigma_O$, and a projection operator $P_O : \Sigma^* \to \Sigma_O^*$ that proceeds by erasing from each word $w \in \Sigma^*$ the letters from Σ_U while preserving all letters from Σ_O in their original order inherited from w. The essence of diagnosability can be described, following [CL08], in this way:

Definition 1 (Diagnosability) *Assume that a DES S has a live (prefix-closed) language $\mathcal{L}(S)$ (i.e. every word $w \in \mathcal{L}(S)$ has some extension $wa \in \mathcal{L}(S)$). Then S is* not *diagnosable w.r.t. unobservable property ϕ iff for every $k \in \mathbb{N}$ there exist $w_n, w_y^k \in \mathcal{L}(S)$ that satisfy*

1. *w_y^k satisfies ϕ ("yes") and w_n ("no") does not;*
2. *w_y^k and w_n are observationally equivalent, i.e. $P_O(w_y^k) = P_O(w_n)$; and*
3. *there is a prefix w_y of w_y^k that satisfies ϕ and such that the suffix of w_y^k* after *w_y has length at least k.*

In other words, the system is *undiagnosable* iff it is ϕ-ambiguous, in the sense that faulty behaviors can remain undistinguishable from healthy behaviors an arbitrary amount of events *after* the condition ϕ arises, e.g. by the occurrence of an unobservable fault event.

Verification. The first standard tool for verifying whether or not a given DES is diagnosable is the *diagnoser* construction found e.g. in [SSL$^+$95, CL08]. Taking as input a finite state machine S with state space X, initial state $x_0 \in X$, transition set $T \subseteq X \times X$ and $\lambda : T \to \Sigma$ with Σ an alphabet that contains the distinguished symbol ε (the *empty word*), define unobservable and observable transition sets T_U and T_O as above, and let $f \in T_U$ be a fault event. Then:

1. take as initial state for the diagnoser the *unobservable reach* of x_0, i.e. the set of all states reachable in S from x_0 without observable events, i.e. only using ε-transitions.
2. Create one state each for
   - every pair (M, y) such that some state $s \in M \subseteq X$ can be reached via some $\bar{w} \in T^*\{f\}T^*$ with $P_O(\bar{w}) = w$, and M is the unobservable reach of s;
   - every pair (M, n) such that some state $s \in X$ can be reached via some $\tilde{w} \in T \backslash \{f\}$, and M is the unobservable reach of s
3. Add transitions to reflect those of S and the propagation of the $y(es)$-label, and the transition from n to y on ocurrence of f.

From this, the *full* set of states of the diagnoser is build following the reachability via transitions and propagating the y-label. One obtains a new automaton - whose size is generally exponential in the size of T - which contains *definite* and *indefinite* states, depending on reachability (in S) with or without faults under the same observation. Diagnosability verification then amounts to checking the obtained automaton for absence of *indefinite* cycles.

This *diagnoser* approach - whose technical details we omit here - had as its main drawback the size of the auxiliary diagnoser structure. The subsequent development of the *verifier* (see below) showed that diagnosability checking is polynomial for untimed automata [YL02], via techniques that can be extended to a polynomial-time diagnosability check for timed DES (Pan and Hashtrudi-Zad [PHZ06]) under a set of non-Zeno type conditions. A noteworthy fact is that the diagnosability problems in both cases reduce to the emptiness problem for (timed) Büchi automata (Cassez [Cas09]).

The *verifier* V associated to a labeled DES S is obtained as the synchronous product two isomorpphic copies S_1 and S_2, i.e. by fusing all pairs $(t_1, t_2) \in T_O^1 \times T_O^2$ having the same (observable) label. All executions of V therefore represent *pairs* of observationnaly behaviours of S, one of which being performed by S_1 and the other by S_2. Denoting f_1 and f_2 the instances of faults in S_1 and S_2, respectively, a violation of diagnosability is witnessed by an *infinite* run of

V on which f_1 occurs but f_2 does not; equivalently, the presence of a *cycle* in V that contains f_1 but not f_2. Let us note in passing that the verifier can be further reduced by surgery that allows to *remove* f_2 whose exploration is not relevant. The verifier construction with its comparatively small size ($|V| = \mathcal{O}(|S|^2)$) is the main reason of polynomiality of diagnosability, and for the hegemonial success in recent years of the verifier (vs diagnoser) method in the literature of the past years.

Extensions. Apart from the extensions to *timed* systems related below, let us indicate that one finds in the literature on DES several equivalent versions, as well as proper *extensions*, of Definition 1.

- In particular, one may let k depend on the choice of w_n; this is equivalent to Definition 1 in the context of finite state systems but no longer when infinite state systems are considered. Another definition that, on the one hand, boils down to Definition 1 in the case of finite automata but is non-equivalent in general takes as witnesses of non-diagnosability two observationally equivalent *infinite* runs.

- **k-Diagnosability:** Conversely, when Definition 1 is modified by *fixing* a value of k, one has the notion of *k-diagnosabiity.* Obviously, k-Diagnosability implies $k+1$-Diagnosability; moreover, Diagnosability in the sense of Definition 1 implies k-diagnosability for all $k > K$ for some threshold $K \in \mathbb{N}$. A finite state system is either $k-$diagnosable for some $k \in \mathbb{N}$ or not diagnosable at all.

- **co-diagnosability:** If several observers are available such that each observer i "sees" a sub alphabet T_i such that $i \neq j$ implies $T_i \cap T_j = \emptyset$, and each observer makes a verdict on f's occurrence, can f be detected by the disjunction of positive verdicts from the different i ?

- **Predictability**: Take for simplicity the case where ϕ is the occurrence of unobservable fault event f. A system is *predictable for f* iff, with the notation of Definition 1, there exists $k \in \mathbb{N}$ with the property that for any faulty word $w_y \in \mathcal{L}(S)$ that decomposes as $w_y = wvf$ with $w \in (\Sigma\backslash\{f\})^*$, $v \in (\Sigma\backslash\{f\})^k$, every $w' \in \mathcal{L}(S)$ such that $P_O(w') = P_O(w)$ has only extensions with a fault in at most k steps, i.e. $w'v' \in \mathcal{L}(s)$ with v' of length at least k implies $v' \notin (\Sigma\backslash\{f\})^*$.

- Active diagnosis [SLT98] aims at improving diagnosability by steering the observed system online, based on the observation thus far, towards behaviour that allows definite diagnosis verdicts. One may view this approach as a dynamic counterpart to works like [CT08b, Cas10] or [PHZ07] in which one aims at minimizing offline the static set of observation sensors needed to ensure diagnosability.

- Other extensions concern stochastic systems, which will be considered later in IMPRO (5.3.); works on timed systems are referred to below.

2.2 Diagnosis and Diagnosability in Timed Systems

We will focus henceforth on the existing approaches for solving these tasks in the context of *timed* DES, starting with diagnosis-related tasks.

- In [BCD05], Bouyer et al consider the problem of diagnosing faults in behaviours of timed plants. We focus on the problem of synthesizing fault diagnosers which are realizable as deterministic timed automata, with the motivation that such diagnosers would function as efficient online fault detectors. We study two classes of such mechanisms, the class of deterministic timed automata (DTA) and the class of event-recording timed automata (ERA). We show that the problem of synthesizing diagnosers in each of these classes is decidable, provided we are given a bound on the resources available to the diagnoser. We prove that under this assumption diagnosability is 2EXPTIME-complete in the case of DTA's whereas it becomes PSPACE-complete for ERA's.

- Like the above, other authors, such as Cassez et al. [Cas09, Cas12, CT08b, Cas10] and Khoumsi et al [KO09], cast the diagnosis and diagnosability problems for *timed* systems in the same framework as those for *untimed* ones. [Cas09] considers algorithms for checking diagnosability of discrete-event systems and timed automata, in both of which cases the diagnosability problem reduces to the emptiness problem for (timed) Büchi automata. Moreover, it is shown that checking whether a timed automaton is diagnosable, can also be reduced to checking bounded diagnosability, as is the case in discrete event systems. [Cas12] show that fault co-diagnosis for both (untimed) finite automata or FA and timed automata (TA) is PSPACE-complete, a result that generalizes to *dynamic* observers in the sense of [CTA07b, CTA07a, CT08b, Cas10, CT08a], i.e. where observation sensors can be switched on and off to minimize resource consumption, and that the codiagnosis problem for TA under bounded resources is 2EXPTIME-complete. Bouyer et al [BCD05] focus on the problem of synthesizing fault diagnosers which are realizable as deterministic timed automata, with the motivation that such diagnosers would function as efficient online fault detectors. two classes of such mechanisms are studied, namely the class of deterministic timed automata (DTA) and the class of event-recording timed automata (ERA). The problem of synthesizing diagnosers in each of these classes is proved decidable, provided one is given a bound on the resources available to the diagnoser. Finally, under this assumption diagnosability is 2EXPTIME-complete in the case of DTA's whereas it becomes PSPACE-complete for ERA's.

- In recent years, Petri net unfoldings have been used to develop diagnosis in concurrent untimed systems, again extending from the untimed case: Benveniste et al [BFHJ03, FBHJ05] use un-timed unfoldings to construct generators explanations for partially observed behaviours; Grabiec et al. [GTJ$^+$10], Jiroveanu et al [BJ13] extend the unfolding based approach to time Petri nets.

- By contrast, Tripakis [Tri02] considers diagnosis of timed automata based on the observation of finite timed sequences. Delays being treated as events, time lapses between (observable) events are observable, increasing discriminating power. Diagnosability is checked in polynomial time via a twin plant construction.

- We mention in passing that a variety of different settings and approaches for particular aspects and applications of real-time supervision exist, such as Meseguer et al. [MPE10] who use *interval* observation in diagnosis ; however, such more remote works are outside the scope of this note and of the project.

2.3 Control of Timed Systems

As in the case of diagnosis, the field of DES control evolved for decades mostly in the *untimed* domain; below we will focus on works that extend to the control of timed systems.

- Maler et al. have introduced the techniques of controller synthesis for real-time systems in [MPS95], with subsequent developments in [AMPS98];

- An alternative semantics for timed games, where delays are strictly coupled with actions, has been proposed by De Alfaro et al. in [dAFH$^+$03];

- An efficient algorithm for solving timed games is proposed in [CDF$^+$05] and a reference implementation is available as [BCD$^+$07];

- Bouyer et al. [BDMP03] consider the problem of synthesizing controllers for timed systems modeled using timed automata, and based on a partial observation. They show that timed control under partial observability is undecidable even for internal specifications (while the analogous problem under complete observability is decidable), and identify a decidable subclass.

- Bouyer et al. [BCL05] use the timed modal logic L_v to specify control objectives for timed plants, and show that the control problem for a large class of objectives can be reduced to a model-checking problem for an extension (L_v^{cont}) of the logic L_v with a new modality. In fact, a fragment of L_v called L_v^{det} is identified, such that any control objective of L_v^{det} can be translated into an L_v^{cont} formula that holds for the plant if and only if there is a controller that can enforce the control objective. The new modality of L_v^{cont} strictly increases the expressive power of L_v, while model-checking of L_v^{cont} remains EXPTIME-complete.

- The problem of minimizing the required observation power, analogous to that of observation minimization for diagnosis in [CTA07b, CTA07a, CT08b, Cas10, CT08a], is addressed in [BCD$^+$12].

- Cassez et al. [CDL+07] consider the problem of controller synthesis for timed games under imperfect information. Strategies must be based on a finite collection of observations and must be stuttering invariant in the sense that repeated identical observations will not change the strategy.
- In Bérard et al. [BHSS12], *concurrent games* on an extension of Vector Addition Systems with States are considered. Inhibition conditions are added for modeling purposes, and asymmetric games are considered; here, both environment and controller have restricted power but do not have the same capabilities. The results include :
  1. reachability games are undecidable for finite targets,
  2. they are 2-EXPTIME-complete for upward-closed targets,
  3. safety games are co-NP-complete for finite, upward-closed and semilinear targets;
  4. moreover, for the decidable cases, a finite representation of the corresponding controllers is constructed.

3 Robustness Issues

3.1 On Robustness of Diagnosis and Diagnosability

When both the observation of a timed discrete event system AND the criteria of diagnosability restricted to its *event labels* and logical time only, the implementation of time observation is not a crucial issue. However, in settings where time enters the definition of the diagnosability problem considered (e.g. is a fault detected at most Δ time units after its occurrence ?), one is led to the problem of observing time in plant monitoring.

Analog vs digital clocks.

Altisen et al. [ACT06][1] study the monitoring and fault-diagnosis problems for dense-time real-time systems, where observers (monitors and diagnosers) have access to digital rather than analog clocks. Analog clocks are infinitely-precise, thus, not implementable : in fact, observation that distinguishes e.g. between an event occurring at any time $t < 1$ and its occurrencence at $t = 1$ can not be implemented by any physical means. Altisen et al. [ACT06] chose an architecture in which a plant is monitored by an observer that masks system events (which do not include time information), and synchronizes with a *digital clock* that produces time "tick" events used to "timestamp" observations. Three problems are considered:

1. given a plant A, a digital clock A_{DC} and a time bound Δ, check whether there exists a diagnoser that can detect any fault within Δ time units after its occurrence;

[1] and independently Kumar et al [XJK10]

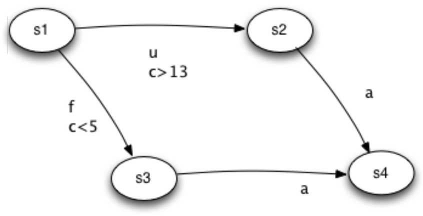


Figure 1: default

2. given A and A_{DC} check whether there is such a diagnoser for some Δ;
3. given A, check whether there is a diagnoser for some digital clock A_{DC} and some Δ.

Given a specification modeled as a timed automaton and a timed automaton model of the digital clock, a sound and optimal (i.e., as precise as possible) digital-clock monitor can be synthesized. Also, given plant and digital clock modeled as timed automata, we can check existence of a digital-clock diagnoser and, if one exists, how to synthesize it. Finally, there are cases where a digital clock, no matter how precise, does not exist, even though the system is diagnosable with analog clocks. Zad et al [ZKW05] treat the clock tick as an extra output signal. In their approach, the required estimates for system condition are updated only when the output changes or when deadlines associated with output changes expire. Thus updates at every clock tick are not required. This in many cases results in reduction in online computing requirements and in the size of the diagnosis system,at the expense of more offline design calculations.

Diagnosis using Time information. The work by Tripakis [Tri02] opens another, broader set of problems. In fact, when delays are considered as events and as objects of observation, a finer and generally faster diagnosis can be attempted. Consider Figure 1; with the conventional naming , u and f are unobservable event labels with f a fault event, and a is an observable label. Starting from $s1$ in time $t = 0$ with clock c at 0, the system can enter state $s4$

- state $s2$ after any timed sequence τ, u, τ, a for $\tau > 13$,
- or state $s3$ after any timed sequence $\tau, f\tau, a$ for $\tau < 5$.

Under the DES scheme of diagnosis, where only events are observed, the system is non-diagnosable since all observations are a. However, in the approach of Tripakis [Tri02], the observation sequence always is τ, a with delay τ observable; that is, different *values* of τ can be distinguished. Hence the diagnoser need only check whether $\tau \in [0, 5[$ - in which case f has *definitely* occurred - or $\tau \in]13, \infty[$ - in which case f *cannot* have occurred. Moreover, a decision on occurrence or non-occurrence of f is available from reading the global time as early as five time units after the start.

Outlook : Robustness Issues in Diagnosability

The sensitivity of timed behaviour impacts the task of diagnosis and diagnosability in several ways; we point out the following:

- In any sense of diagnosis, and beyond the choice of robustly implementable *architectures* such as the *digital clock* setup above, one should aim at understanding *robust* diagnosability in a broader mathematical form. The intuition is that "small enough" perturbations of the time parameters of system S must preserve its diagnosability (and/or the diagnosis *verdicts*). In other words, S must be embedded in a diagnosable neighborhood w.r.t. an adequate metric topology of systems; the task is to identify conditions under which these embeddings are available.

- In the context of *observable time delays* as in Tripakis [Tri02], the subtleties of diagnosis need to be explored more in depth. In particular, it needs to be checked whether diagnosability in this sense is, or can be made, robust under perturbations in the temporal parameters.

3.2 Synthesis of robust systems

Checking that a system is robust or implementable is one (already difficult) step in the verification process. However if one finds that the system is actually not robust, it is difficult to correct it. So instead of building a system and verifying *a posteriori* that it is robust w.r.t. its specification, it would be more efficient to start with a specification of the system and to directly synthesize a model that is robustly correct by construction.

The leads to be followed are thus:

- Synthesize systems that are robustly correct, either independently of *any* control or such that control can be exerted efficiently and with guarantees of robust reaction from the system. Here again, the use of powerful mathematical frameworks appears crucial.

- In the line of [XK10] and extending the anlogours work on diagnosis from [ACT06, XJK10], study controller synthesis for discrete-event systems augmented by a digital clock.

- Explore *properly* time-related properties of the system behaviour in devising control strategies that exploit the effect of *delays* (compare the toy example in Figure 1) rather than observable *events* to achieve control goals.

4 Contributions from the ImpRo project

We include two papers written within the project in this deliverable, and addressing some of the issues outlined in the previous sections.

In [JFLR12], we investigate the issue of the control of parametric timed systems. In this setting, parameters are used to represent durations, which could naturally account for the uncertainties on delays introduced by, e.g., an implementation on some specific hardware. We focus on reachability control objectives but many interesting properties can be reduced to that setting, in particular simulation / refinement verification. In the paper, we introduce decidable classes for the existence of parameter valuations satisfying some reachability property and provide a semi-algorithm, extending the one in [CDF$^+$05].

In [BMS12], we solve a robust reachability problem using the framework of timed games: we are interested in synthesizing "robust" strategies for ensuring reachability of a location in a timed automaton; with "robust", we mean that it must still ensure reachability even when the delays are perturbed by the environment. We model this perturbed semantics as a game between the controller and its environment, and solve the parameterized robust reachability problem: we show that the existence of an upper bound on the perturbations under which there is a strategy reaching a target location is EXPTIME-complete.

References

[ACT06] Karine Altisen, Franck Cassez, and Stavros Tripakis. Monitoring and fault-diagnosis with digital clocks. In *6th Int. Conf. on Application of Concurrency to System Design (ACSD'06)*. IEEE Computer Society, June 2006.

[AMPS98] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller Synthesis for Timed Automata. In *Proc. IFAC Symp. on System Structure & Control*, pages 469–474. Elsevier Science, 1998.

[BCD05] Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. Fault diagnosis using timed automata. In Vladimiro Sassone, editor, *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 219–233, Edinburgh, Scotland, UK, April 2005. Springer.

[BCD$^+$07] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Uppaal-tiga: Time for playing games! In *19th International Conference on Computer Aided Verification (CAV 2007)*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125, Berlin, Germany, July 2007. Springer.

[BCD$^+$12] P. Bulychev, F. Cassez, A. David, K. G. Larsen, J-F. Raskin, and P-A. Reynier. Controllers with Minimal Observation Power (Application to Timed Systems). In *Proc. ATVA 2012*, 2012.

[BCL05] Patricia Bouyer, Franck Cassez, and François Laroussinie. Modal logics for timed control. In Martín Abadi and Luca de Alfaro, editors, *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 81–94, San Francisco, CA, USA, August 2005. Springer.

[BDMP03] Patricia Bouyer, Deepak D'Souza, P. Madhusudan, and Antoine Petit. Timed control with partial observability. In Warren A. Hunt, Jr and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192, Boulder, Colorado, USA, July 2003. Springer.

[BFHJ03] Albert Benveniste, Éric Fabre, Stefan Haar, and Claude Jard. Diagnosis of asynchronous discrete event systems: A net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, May 2003.

[BHSS12] Béatrice Bérard, Serge Haddad, Mathieu Sassolas, and Nathalie Sznajder. Concurrent games on VASS with inhibition. In Maciej Koutny and Irek Ulidowski, editors, *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, pages 39–52, Newcastle, UK, September 2012. Springer.

[BJ13] René. Boel and George Jiroveanu. The on-line diagnosis of time petri nets. In Carla Seatzu, Manuel Silva, and Jan H. van Schuppen, editors, *Control of Discrete-Event Systems*, volume 433 of *Lecture Notes in Control and Information Sciences*, pages 343–364. Springer London, 2013.

[BMS12] Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust reachability in timed automata: A game-based approach. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP'12) – Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 128–140, Warwick, UK, July 2012. Springer.

[Cas09] Franck Cassez. A Note on Fault Diagnosis Algorithms. In *48th IEEE Conference on Decision and Control and 28th Chinese Control Conference*, Shanghai, P.R. China, December 2009. IEEE Computer Society. Extended Version.

[Cas10] Franck Cassez. Dynamic Observers for Fault Diagnosis of Timed Systems. In *49th IEEE Conference on Decision and Control and 28th Chinese Control Conference*, Atlanta, December 2010. IEEE Computer Society.

[Cas12] Franck Cassez. The Complexity of Codiagnosability for Discrete Event and Timed Systems. *IEEE Transactions on Automatic Control*, 2012. Forthcoming.

[CDF$^+$05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *16th International Conference on Concurrency Theory (CONCUR 2005)*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80, San Fransisco, CA, USA, August 2005. Springer-Verlag.

[CDL$^+$07] Franck Cassez, Alexandre David, Kim G. Larsen, Didier Lime, and Jean-François Raskin. Timed control with observation based and stuttering invariant strategies. In *5th Int. Symp. on Automated Technology for Verification and Analysis (ATVA'07)*, volume 4762 of *Lecture Notes in Computer Science*, pages 307–321. Copyright *Springer*, October 2007.

[CL08] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems (2nd edition)*. Springer, 2008.

[CT08a] Franck Cassez and Stavros Tripakis. Fault diagnosis with dynamic diagnosers. In *Proc. of the* 9th *Workshop on Discrete Event Systems (WODES'08)*, pages 212–217. IEEE Computer Society, May 2008.

[CT08b] Franck Cassez and Stavros Tripakis. Fault diagnosis with static and dynamic diagnosers. *Fundamenta Informaticae*, 88(4):497–540, November 2008.

[CTA07a] Franck Cassez, Stavros Tripakis, and Karine Altisen. Sensor minimization problems with static or dynamic observers for fault diagnosis. In *7th Int. Conf. on Application of Concurrency to System Design (ACSD'07)*, pages 90–99. IEEE Computer Society, July 2007.

[CTA07b] Franck Cassez, Stavros Tripakis, and Karine Altisen. Synthesis of optimal dynamic observers for fault diagnosis of discrete-event systems. In *1st IEEE & IFIP Int. Symp. on Theoretical Aspects of Soft. Engineering (TASE'07)*, pages 316–325. IEEE Computer Society, June 2007.

[dAFH$^+$03] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In Roberto M. Amadio and Denis Lugiez, editors, *14th International Conference on Concurrency Theory (CONCUR 2003)*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156, Marseilles, France, 2003. Springer.

[FBHJ05] Éric Fabre, Albert Benveniste, Stefan Haar, and Claude Jard. Distributed monitoring of concurrent and asynchronous systems. *Discrete Event Dynamic Systems: Theory and Applications*, 15(1):33–84, March 2005.

[GTJ$^+$10] Bartosz Grabiec, Louis-Marie Traonouez, Claude Jard, Didier Lime, and Olivier H. Roux. Diagnosis using unfoldings of parametric time petri nets. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *FORMATS*, volume 6246 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 2010.

[JFLR12] Aleksandra Jovanović, Sébastien Faucou, Didier Lime, and Olivier H. Roux. Real-time control with parametric timed reachability games. In *11th International Workshop on Discrete Event Systems (WODES'12)*, pages 323–330, Guadalajara, Mexico, October 2012. IFAC.

[KO09] A. Khoumsi and L. Ouédraogo. Diagnosis of faults in real-time discrete event systems. In *Proceedings of the IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SafeProcess), Barcelona/Spain, July 2009*, 2009.

[MPE10] Jordi Meseguer, Vicen Puig, and Teresa Escobet. Fault diagnosis using a timed discrete-event approach based on interval observers: Application to sewer networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, pages 900–916, 2010.

[MPS95] O. Maler, A. Pnueli, and J. Sifakis. On the Synthesis of Discrete Controllers for Timed Systems. In *Proc. 12th Symp. on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *LNCS*, pages 229–242. Springer, 1995.

[PHZ06] J. Pan and S. Hashtrudi-Zad. Diagnosability test for timed discrete event systems. In *Proc. 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 06)*, pages 63–72, Washington, DC, USA, November 2006.

[PHZ07] J. Pan and S. Hashtrudi-Zad. Diagnosability analysis and sensor selection in discrete event systems with permanent failures. In *Proc. 3rd annual IEEE Conference on Automation Science and Engineering (IEEE CASE 2007)*, pages 869–874, Scottsdale, Arizona, USA, September 2007.

[SLT98] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete event systems. *IEEE Transactions on Automatic Control*, 43(7):908–929, July 1998.

[SSL$^+$95] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[Tri02] S. Tripakis. Fault diagnosis for timed automata. In *Proceedings FTRTFT'02*, 2002.

[XJK10] S. Xu, S. Jiang, and R. Kumar. Diagnosis of dense-time systems using digital clocks. *IEEE Transactions on Automation Science and Engineering*, 7(4):870–878, 2010.

[XK10] S. Xu and R. Kumar. Real-time supervisory control of discrete event systems using digital- clocks. *IEEE Transactions on Automatic Control*, 55(9):2003–2013, 2010.

[YL02] T-S. Yoo and S. Lafortune. Polynomial time verification for diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 47(9):1491–1495, 2002.

[ZKW05] S. Hashtrudi Zad, R.H. Kwong, and W.M. Wonham. Fault diagnosis in discrete-event systems: Incorporating timing information. *IEEE Transactions on Automatic Control*, 50(7):1010–1015, July 2005.

Real-Time Control with Parametric Timed Reachability Games [★]

A. Jovanović [**] S. Faucou [***] D. Lime [**] O. H. Roux [**]

LUNAM Université.
[**] École Centrale de Nantes (e-mail: aleksandra.jovanovic@irccyn.ec-nantes.fr, didier.lime@irccyn.ec-nantes.fr, olivier-h.roux@irccyn.ec-nantes.fr).
[***] Université de Nantes (e-mail: sebastien.faucou@univ-nantes.fr)
IRCCyN UMR CNRS 6597

Abstract: Timed game automata are used for solving control problems on real-time systems. A timed reachability game consists in finding a strategy for the controller for the system, modeled as a timed automaton. Such a controller says when and which of some "controllable" actions should be taken in order to reach "goal" states. We deal with a parametric version of timed game automata. We define parametric timed reachability games and introduce their subclass for which the existence of a parameter valuation, such that there is a strategy for the controller to reach the "goal" state, is decidable. We also propose a semi-algorithm to symbolically compute the corresponding set of parameter valuations.

Keywords: Timed automata, game theory, parameters, control, verification, model-checking

1. INTRODUCTION

Formal methods are widely used in the analysis of time critical systems. For instance, methods such as model-checking allow the verification of a system by exploring the state-space of a (timed) model, e.g. a timed automaton. A prerequisite for these methods is the availability of a complete model of the system. Thus, it can be difficult to use them at early design stages, when the whole system is not fully characterized.

It is sometimes possible to overcome this problem by using parameters for modeling values that are not fully characterized yet. To exploit such models, a parametric approach in automata theory must be used. The analysis of a parametric model produces symbolic constraints on the parameters that ensure the correctness of the system.

Parametric control problem The *verification* problem for a given model of the system S and a model of the specification φ, consists in checking whether S satisfies φ, which is often written $S \models \varphi$ and referred to as the *model-checking problem.* The *parametric model-checking problem* for a parametric system S and a parametric specification φ, consists in checking whether there exists a valuation v of the parameters such that S satisfies φ for this valuation, which is written $[\![S]\!]_v \models [\![\varphi]\!]_v$.

The *control problem* assumes the system is *open* i.e. we can restrict the behaviour of S: some events in S are *controllable* and the others are *uncontrollable*, and we can sometimes disable controllable actions. The *control problem* for a system S and a specification φ asks the following: is there a controller C s.t. $S \times C \models \varphi$? The *parametric control problem* for a parametric system S and a parametric specification φ asks the following: is there a controller C and a valuation v of the parameters s.t. $[\![S]\!]_v \times C \models [\![\varphi]\!]_v$? The associated *parametric controler synthesis problem* asks to compute a witness controller C and the set of valuations V such that $\forall v \in V$, $[\![S]\!]_v \times C \models [\![\varphi]\!]_v$.

The control problem can be formulated as a game in which the controller plays against the environment.

Related Work Parametric timed automaton (PTA) has been introduced in Alur et al. (1993) as an extension of the timed automaton, Alur and Dill (1994), that allows the use of parameters instead of concrete values in the clock constraints. The emptiness problem for PTA asks: is the set of parameter valuations, such that an automaton has an accepting run, empty? The main result is that this problem is undecidable in general except for some restricted cases. In Hune et al. (2002), the authors introduce a subclass of parametric timed automata, called lower bound/upper bound (L/U) automata, for which the emptiness problem is decidable.

Timed game automata (TGA), Maler et al. (1995), were introduced for solving control problems on real-time systems. TGA are based on the framework of Ramadge and Wonham (1989), developed for the control of the discrete event processes. A timed game automaton is essentially a timed automaton whose set of actions is partitioned into controllable and uncontrollable actions. Two players, a controller and an environment, choose at every instant one of the available actions from their own sets and the game progresses. A *timed reachability game* consists in finding a strategy for the controller: when and which of the controllable actions should be taken such that, regardless of what the environment does, the system ends up in

[★] This work was supported by project ANR-2010-BLAN-0317.

a desired location. Such timed games are known to be decidable, Maler et al. (1995); Asarin et al. (1998).

Our Contribution We first define parametric timed game automata (PGA) and introduce their subclasses for which we prove the decidability of the emptiness problem for parametric timed reachability game i.e. the existence of a parameter valuation, such that there is a strategy for the controller to reach the "goal" state. We then propose the extension for the parametric case of the algorithm of Cassez et al. (2005) for solving timed games. It leads to a semi-algorithm that symbolically computes the set of corresponding parameter valuations. We end with a case study that illustrates the use of the new subclass and proposed algorithm.

2. PARAMETRIC TIMED GAMES

2.1 Basic Definitions

Parametric constraints Let $X = \{x_1, ...x_m\}$ be a finite set of variables modeling *clocks*. $\mathbb{R}$ is the set of real numbers, $\mathbb{Q}$ the set of rational numbers, and $\mathbb{Z}$ the set of integers. A *clock valuation* is a function $w : X \mapsto \mathbb{R}_{\geq 0}$ assigning a non-negative real value to each clock. Let $P = \{p_1, ...p_n\}$ be a finite set of *parameters*. A *parametric clock constraint* c is an expression of the form $c ::= x_i - x_j \backsim p \mid x_i \backsim p \mid c \wedge c$, where $x_i, x_j \in X$, $\backsim \in \{\leq, <\}$, and p is a linear expression of the form $k_0 + k_1 p_1 + ... + k_n p_n$ with $k_0, ...k_n \in \mathbb{Z}$. A *parameter valuation* is a function $v : P \mapsto \mathbb{Q}$ assigning a rational value to each parameter. For any parametric clock constraint c and any parameter valuation v, we note $[\![c]\!]_v$ the constraint obtained by replacing each parameter p_i by its valuation $v(p_i)$. A pair (w, v) of a clock valuation and a parameter valuation satisfies a parametric constraint c, notation $(w, v) \models c$, if the expression $c[w, v]$, obtained by replacing each parameter p_i with $v(p_i)$ and each clock x_i with $w(x_i)$, evaluates to true. We denote by $G(X)$ the set of parametric constraints over X, and $G'(X)$ a set of parametric constraints over X of the form $c' ::= x_i \backsim p \mid c' \wedge c'$. If w is a clock valuation and $t \in \mathbb{R}_{\geq 0}$, we write $w + t$ for the valuation assigning $w(x) + t$ to each clock. For $R \subseteq X$, $w[R]$ denotes a valuation assigning 0 to each clock in R and $w(x)$ to each clock in $X \backslash R$.

Note that given an arbitrary order on $X \cup P$, any valuation (w, v) can be identified to a point in $\mathbb{R}^{|P|+|X|}$, and the set of valuations (w, v) such that some parametric constraint c is true is then a convex polyhedron in that space.

Parametric Timed Automata Parametric clock constraints are used as guards and invariants in parametric timed automata.

Definition 1. A Parametric Timed Automaton (PTA) is a tuple $\mathcal{A} = (L, l_0, X, \Sigma, P, E, I)$, where L is a finite set of locations, $l_0 \in L$ is an initial location, X is a finite set of clocks, Σ is a finite alphabet of actions, P is a finite set of parameters, $E \subseteq L \times \Sigma \times G(X) \times 2^X \times L$ is a finite set of transitions, and $I : L \mapsto G'(X)$ is a function that assigns an invariant to each location.

Definition 2. (Semantics of a PTA). The concrete semantics of a PTA $\mathcal{A}$ under a parameter valuation v, notation $[\![\mathcal{A}]\!]_v$, is the labeled transition system $(Q, q_0, \rightarrow)$ over $\Sigma \cup \mathbb{R}_{\geq 0}$ where:

- $Q = \{(l, w) \in L \times (X \mapsto \mathbb{R}_{\geq 0}) \mid (w, v) \models I(l)\}$
- $q_0 = \{(l, w_0) \in Q \mid l = l_0 \wedge w_0 : X \mapsto 0\}$
- the transition relation $\rightarrow$ is defined as: $\forall (l, w), (l', w') \in Q$, $t \geq 0$ and $a \in \Sigma$:

  delay transition: $(l, w) \xrightarrow{t} (l, w')$ if $w' = w + t$

  action transition: $(l, w) \xrightarrow{a} (l', w')$ if $\exists g \in G(X)$, $R \subseteq X : l \xrightarrow{a,g,R} l'$ and $(w, v) \models g$ and $w' = w[R]$

A run of a parametric timed automaton $\mathcal{A}$ under a parameter valuation v, is a sequence of alternating delay and action transitions in the semantics of $[\![\mathcal{A}]\!]_v$. $Runs((l, w), [\![\mathcal{A}]\!]_v)$ denotes the set of runs starting in (l, w), and $Runs([\![\mathcal{A}]\!]_v)$ denotes the set of runs starting in the initial state (l_0, w_0). If ρ is a finite run, we denote by $last(\rho)$ the last state of ρ.

A state (l, w) is said to be *reachable* if there exists a finite run $\rho \in Runs([\![\mathcal{A}]\!]_v)$ such that $last(\rho) = (l, w)$.

L/U Automata A parameter p_i occurs positively (resp. negatively) in a parametric constraint $x_i - x_j \sim k_0 + k_1 p_1 + ... + k_n p_n$, with $\sim \in \{\leq, <\}$, if $k_i > 0$ (resp. $k_i < 0$). A *lower bound* (resp. *upper bound*) parameter of a parametric timed automaton $\mathcal{A}$ is a parameter that only occurs negatively (resp. positively) in the parametric constraints of $\mathcal{A}$.

Definition 3. (L/U automaton Hune et al. (2002)). A L/U automaton is a parametric timed automaton where every parameter is either a lower or an upper bound parameter.

Let $\mathcal{A}$ be a non-parametric timed automaton. Weakening the guards and invariants in $\mathcal{A}$ (decreasing the lower bounds and increasing the upper bounds on clocks) yields an automaton whose reachable states include those of $\mathcal{A}$, and strengthening the guards and invariants in $\mathcal{A}$ (increasing the lower bounds and decreasing the upper bounds on clocks) yields an automaton whose reachable states are a subset of those of $\mathcal{A}$. Following this fact, decidability of the reachability-emptiness problem (existence of a parameter valuation such that a certain state in the automaton is reachable from the initial state) has been established for L/U automata (detailed proof in Hune et al. (2002)).

2.2 Parametric Timed Games

We now extend the previous definitions to obtain a more powerful formalism that allows us to express control problems. To this end we will parametrize the classical notion of timed games.

Definition 4. A Parametric (Timed) Game Automaton (PGA) $\mathcal{G}$ is a parametric timed automaton with its set of actions Σ partitioned into controllable (Σ^c) and uncontrollable (Σ^u) actions.

As for PTA, for any PGA $\mathcal{G}$ and any parameter valuation v, we obtain a timed game automaton $[\![\mathcal{G}]\!]_v$ by replacing each parametric constraint c in the guards and invariants by $[\![c]\!]_v$.

In a timed game, each of the two players acts within its set of actions according to a *strategy*. The game being symmetric, we only give the definitions for Player 1 (the controller playing with actions in Σ^c). Since we consider only reachability properties, as in Maler et al. (1995); Cassez et al. (2005), invariants restrict the behavior of the automaton but never force a player to move.

Definition 5. (Strategy). A strategy $\mathcal{F}$ over $[\![\mathcal{G}]\!]_v$ is a partial function from $Runs([\![\mathcal{G}]\!]_v)$ to $\Sigma^c \cup \{\mathsf{delay}\}$ such that for every finite run ρ, if $\mathcal{F}(\rho) \in \Sigma^c$ then $last(\rho) \xrightarrow{\mathcal{F}(\rho)} q$ for some state $q = (l, w)$, and if $\mathcal{F}(\rho) = \mathsf{delay}$, then there exists some $d > 0$ such that for all $0 \leq d' \leq d$, there exists some state q such that $last(\rho) \xrightarrow{d'} q$.

A strategy therefore tells each player, given the history of the game, whether to play one of its actions or to wait (delay). We consider only memory-less strategies, where $\mathcal{F}(\rho)$ only depends on the current state $last(\rho)$, and which are sufficient for timed reachability games, Asarin et al. (1998), and, as we will show, for our parametric extension.

It should be noted that the uncontrollable actions cannot be relied on to reach the desired state: the controller has to be able to reach the desired locations by itself.

An example, shown in Figure 2, and a case-study, presented in Section 5, illustrate the strategy to reach a desired location.

The restricted behaviour of $[\![\mathcal{G}]\!]_v$ when the controller plays the strategy $\mathcal{F}$ against all possible strategies of the environment is defined by the notion of *outcome*.

Definition 6. (Outcome). Let $\mathcal{G}$ be a PGA, v be a parameter valuation, and $\mathcal{F}$ be a strategy over $[\![\mathcal{G}]\!]_v$. The outcome $Outcome(q, \mathcal{F})$ of $\mathcal{F}$ from state q is the subset of finite runs in $Runs(q, [\![\mathcal{G}]\!]_v)$ defined inductively as:

- the run with no action $q \in Outcome(q, \mathcal{F})$
- if $\rho \in Outcome(q, \mathcal{F})$ then $\rho' = \rho \xrightarrow{\delta} q' \in Outcome(q, \mathcal{F})$ if $\rho' \in Runs(q, [\![\mathcal{G}]\!]_v)$ and one of the following three condition holds:
  (1) $\delta \in \Sigma^u$,
  (2) $\delta \in \Sigma^c$ and $\delta = \mathcal{F}(\rho)$,
  (3) $\delta \in \mathbb{R}_{\geq 0}$ and $\forall 0 \leq \delta' < \delta$, $\exists q'' \in S$ s.t. $last(\rho) \xrightarrow{\delta'} q'' \wedge \mathcal{F}(\rho \xrightarrow{\delta'} q'') = \mathsf{delay}$.
- for an infinite run ρ, $\rho \in Outcome(q, \mathcal{F})$, if all the finite prefixes of ρ are in $Outcome(q, \mathcal{F})$.

As we are interested in reachability games, we want to consider only the runs in the outcome that are "long enough" to have a chance to reach the goal: a run $\rho \in Outcome(q, \mathcal{F})$ is *maximal* if it is either infinite or there is no delay d and no state q' such that $\rho' = \rho \xrightarrow{d} q' \in Outcome(q, \mathcal{F})$ and $\mathcal{F}(\rho') \in \Sigma^c$ (the only possible actions from $last(\rho)$ are uncontrollable actions), or ρ is an infinite run. We note $MaxOut(q, \mathcal{F})$ the set of maximal runs for state s and strategy $\mathcal{F}$.

We can now define the notion of *winning strategy*.

Definition 7. (Winning strategy). Let $\mathcal{G} = (L, l_0, X, \Sigma^c \cup \Sigma^u, P, E, I)$ be a PGA and $\mathsf{goal} \in L$. A strategy $\mathcal{F}$ is winning for the location goal if for all runs $\rho \in MaxOut(q_0, \mathcal{F})$, where $q_0 = (l_0, w_0)$, there is some state (goal, w) in ρ.

Similarly, a state s is winning (for the controller) if it belongs to a run in the outcome of a winning strategy.

In the parametric case, non-existance of a winning strategy can be solved by the modification of a model, as illustrated in the case-study, Section 5.

Definition 8. Emptiness problem for parametric timed reachability game for PGA is the problem of determining whether the set of parameter valuation, such that there is a strategy for the controller to reach the desired state, is empty.

The emptiness problem for PTA is known to be undecidable, Alur and Dill (1994). As PGA extend PTA, the following theorem stands.

Theorem 1. The emptiness problem for parametric timed reachability game for PGA is undecidable.

We accordingly extend the subclass of L/U automata of Hune et al. (2002) to define a subclass of parametric game automata for which this problem is decidable.

3. L/U REACHABILITY TIMED GAMES

The parameters are partitioned into two sets. The first set P^l contains parameters that are used as lower bounds in the guards on the controllable transitions and as upper bounds in the guards of the uncontrollable transitions. The parameters from the other set, P^u, are used as upper bounds in the controllable and lower bounds in the uncontrollable transitions. This is a natural way of making the controller more powerful by restricting possible behaviors of the environment (and vice-versa). We assume that invariants are non-parametric constraints.

Definition 9. (L/U game automata). A L/U game automaton $\mathcal{G} = (L, l_0, X, \Sigma^c \cup \Sigma^u, P, E, I)$ is a parametric game automaton in which:

- the set of parameters P is partitioned as P^l and P^u;
- each parameter $p \in P^l$ occurs only negatively (resp. positively) in the guards of controllable (resp. uncontrollable) transitions;
- each parameter $p \in P^u$ occurs only positively (resp. negatively) in the guards of controllable (resp. uncontrollable) transitions;
- for each location l, $I(l)$ contains no parameter.

We will now prove that the existence of a parameter valuation, such that the controller has a winning strategy, is decidable for L/U games.

Let (λ, μ) represent a parameter valuation such that λ applies to lower bound parameters, and μ applies to upper bound parameters. Let $\mathcal{G}[(0, \infty)]$ represent the (extended) parameter valuation of $\mathcal{G}$ such that each parameter $p^u_i \in P^u$ is set to ∞, and each parameter $p^l_i \in P^l$ is set to 0. In this way we obtain a timed game automaton, for which the existence of a winning strategy is known to be decidable.

Lemma 1. Let $\mathcal{G}$ be a L/U game automaton. There exists a parameter valuation (λ, μ) such that a goal state is enforceable in $[\![\mathcal{G}]\!]_{(\lambda,\mu)}$ with a strategy $\mathcal{F}$, if and only if a goal state is enforceable in $\mathcal{G}[(0, \infty)]$ using the same strategy $\mathcal{F}$.

Proof. We are searching for the value for upper bound parameters such that all runs of a given strategy $\mathcal{F}$ remain winning. For lower bound parameters, zero valuation is a possible solution. Note that each run of a *winning* strategy necessarily reaches a goal state in finite time and with a finite number of discrete actions.

To find the upper bound value, we introduce in the system a new clock x_0 that serves only to measure the time elapsed from the start. Consider first the untimed runs of $\mathcal{F}$. Since there is a finite number of edges, there is a finite number of possible untimed runs. For each of those runs we measure its *sufficient* duration in a timed case of $\mathcal{G}[(0,\infty)]$. A sufficient duration is obtained when the uncontrollable transitions are taken as late as possible in each state (which means just before we should take a supposed controllable transition in that state) and the controllable transitions are taken when $\mathcal{F}$ says. Since $\mathcal{F}$ is winning, each run i will reach a goal location at some time $x_0^i = T_i$.

We take the maximal value, T_i^{max}, for the upper bound parameters, and 0 for the lower bound parameters. In this valuation, no matter what the environment does, all guards will be satisfied so that we can take the controllable actions at the supposed time. Hence, all runs of the strategy $\mathcal{F}$ remain winning. $\square$

Based on Lemma 1, we can formulate the following theorem.

Theorem 2. The emptiness problem for parametric timed reachability game for L/U game automata is decidable.

If we think in terms of control it may not be very realistic to be allowed to forbid uncontrollable transitions using the values of their parameters. It may actually even seem a bit far-fetched to parametrize uncontrollable actions at all. A consequence of the previous result however, is that for the subclass of L/U game automata with no parameter in the guards of uncontrollable transitions, the problem of the emptiness of the set of valuations such that the controller has a winning strategy is decidable too.

In the context of games however, having a game as symmetric as possible, including parametrization of guards makes sense. We will now explore the case in which we nonetheless impose that the parameter valuations never set the guards of the uncontrollable transitions to false: we can restrict their behaviour but not to the point of uniformly forbidding the transition.

Consequently, all the guards on the uncontrollable transitions that contain a parameter as a lower (resp. upper) bound have to contain a constant as a non-strict upper (resp. lower) bound. Non-strict inequalities are mandatory so that a clock can take the value equal to the constant as a single time point in the emptiness test. The guards on the controllable transitions have no other restriction than the L/U condition.

We also limit the parametric linear expression in the constraints of uncontrollable transitions to just one parameter.

Definition 10. (Restricted L/U game automata). A restricted L/U game automaton is a L/U game automaton in which the guards of the uncontrollable transitions are constraints of the form $k \leq x \leq Ka$ or $Ka \leq x \leq k$, where $x \in X$, $a \in P$, $k \in \mathbb{Q}$ and $K \in \mathbb{Z}$.

We will now establish the decidability of the emptiness problem for this subclass of L/U game automata. Recall that the parameters $p_i^u \in P^u$ (resp. $p_i^l \in P^l$) are used as the lower (resp. upper) bounds in the guards on the uncontrollable transitions. Let $min(p_i^u)$ be the minimal constant that appears as an upper bound in the guards containing p_i^u as a lower bound, and $max(p_i^l)$ be a maximal constant that appears as a lower bound in the guards containing p_i^l as an upper bound. Let $\mathcal{G}_r[max, min]$ represent a valuation of $\mathcal{G}$ that assign 0 (resp. ∞) to each lower (resp. upper) bound parameter that appears only in controllable transitions, and $max(p_i^l)$ (resp. $min(p_i^u)$) to every other p_i^l (resp. p_i^u) bound parameter.

Lemma 2. Let $\mathcal{G}_r$ be a restricted L/U game automaton. There is a parameter valuation (λ, μ) such that a goal state is enforceable in $[\![\mathcal{G}_r]\!]_{(\lambda,\mu)}$ with a strategy $\mathcal{F}$, if and only if the goal state is enforceable in $\mathcal{G}_r[max, min]$ using the same strategy $\mathcal{F}$.

Proof. Again we look for the value for upper bound parameters appearing only in the guards on controllable transitions. Parameter valuations (λ, μ), that assign values $max(p_i^l)$ and $min(p_i^u)$ to parameters p_i^l and p_i^u, that do not appear in controllable transition guards, is a solution. For those parameters, p_k^l and p_k^u, that appear only in guards of controllable transitions, valuations (λ, μ) assign 0 and T_i^{max}, respectively, where the value T_i^{max} is obtained from $\mathcal{G}_r[max, min]$ as in the proof for Lemma 1. $\square$

Similarly, the next theorem follows Lemma 2.

Theorem 3. The emptiness problem for parametric timed reachability game for restricted L/U game automata is decidable.

The use of L/U game automata is shown in the case-study, 5, presenting a copper annealing controller.

4. SYMBOLIC STATE-SPACE EXPLORATION

For timed reachability games, a strategy for the controller is synthesized using the backwards algorithm for solving timed games, Maler et al. (1995). We now define the needed operations.

Let $X \subseteq Q$ and $a \in \Sigma$. The action predecessor of X, $Pred_a(X) = \{(l,w) \mid \exists (l',w') \in X, (l,w) \xrightarrow{a} (l',w')\}$, is extended for the controllable and uncontrollable action predecessors of X in a timed game automaton: $cPred(X) = \bigcup_{c\in\Sigma^c} Pred_c(X)$ and $uPred(X) = \bigcup_{u\in\Sigma^u} Pred_u(X)$, respectively. The action successor is defined as follows $Post_a(X) = \{(l',w') \mid \exists (l,w) \in X, (l,w) \xrightarrow{a} (l',w')\}$.

Timed successors and timed predecessors of X are defined by $X \nearrow = \{(l, w+d) \mid (l,w) \in X \wedge (w+d) \models I(l)\}$, $X \swarrow = \{(l, w-d) \mid (l,w) \in X\}$, respectively.

We also define a *safe-timed predecessors* operator $Pred_t(X_1, X_2)$. A state (l,w) is in $Pred_t(X_1, X_2)$ if from (l,w) we can reach $(l',w') \in X_1$ by time elapsing and along the path from (l,w) to (l',w') avoid X_2, formally:

$Pred_t(X_1, X_2) = \{q' \in Q \mid \exists d \in \mathbb{R}_{\geq 0} \text{ s.t. } q \xrightarrow{d} q', q' \in X_1 \text{ and } Post_{[0,d]}(q) \subseteq Q \backslash X_2\}$, where $Post_{[0,d]}(q) = \{q' \in Q \mid \exists t \in [0,d] \text{ s.t. } q \xrightarrow{t} q'\}$

The *controllable predecessors* operator is defined as follow:

$$\pi(X) = Pred_t(X \cup cPred(X), uPred(X))$$

In practice, the analysis of timed automata is based on the exploration of a finite graph, the *simulation graph*. The nodes of the simulation graph are symbolic states $S = (l, Z)$, where $l \in L$ and Z is a subset of a clock-space $\mathbb{R}^X_{\geq 0}$ defined by a clock constraint, called *zone*. Relation $\longrightarrow$ defines the edges of the simulation graph as $(l, Z) \xrightarrow{a} (l', Z')$, if there is a transition $(l, a, g, R, l') \in E$, and Z' is a zone successor by an edge of Z.

4.1 Algorithm for Solving Timed Games Cassez et al. (2005)

In Cassez et al. (2005), the authors present a symbolic on-the-fly algorithm for solving timed reachability games, which is based on the simulation graph and given in Figure 1. This algorithm consists of a forward computation of the simulation graph and a backward propagation of information of winning states (the states from which there is a strategy to reach the goal location). The *Waiting* set represents a list of edges in the simulation graph waiting to be explored. The *Past* set contains all symbolic states that have already been encountered. The winning status of a symbolic state S is represented by $Win[S]$, a subset of the symbolic state S which is currently known to be winning. The dependency set of S, $Depend[S]$, contains a set of edges (predecessors of S), which must be re-added to *Waiting* set when a new information about $Win[S]$ is obtained. Whenever an edge $e = (S, \alpha, S')$ is considered with S' belonging to *Passed*, it is added to a dependency set of S' in order that a possible future information about additional winning states of S' may be back propagated to S. Since zones are used as underlying data structure, all the steps of the algorithm are carried out efficiently. Finally, upon the termination of the algorithm, set Win^* contains all the winning states of the simulation graph.

4.2 Extension for parameter synthesis

The algorithm of Cassez et al. (2005), being on-the-fly, stops as soon as the initial state becomes winning. In the parametric case we are rather interested in computing all the conditions on parameters such that there exists a winning strategy to reach the goal state. That is why our extension of the algorithm removes the condition (of the while loop) $S_0 \in Win[S_0]$, and it is not on-the-fly. Also, we have chosen to "parametrize" the algorithm of Cassez et al. (2005), instead of the plain backwards fixpoint computation of Maler et al. (1995), because the added forward exploration pre-constrains the parameters, with conditions allowing for the reachability of symbolic states, and therefore makes the backward propagation more efficient.

Parametric Symbolic States To modify this algorithm so as to compute parameter valuations, we use an extended notion of symbolic state in which we have a location and a *parametric zone* Z - a polyhedron constraining both clocks and parameters together, i.e., a set of pairs (w, v) satisfying a parametric clock constraint.

```
Initialization:
$Passed \leftarrow \{S_0\}$ where $S_0 = \{(l_0, \mathbf{0})\}\nearrow$;
$Waiting \leftarrow \{(S_0, \alpha, S') \mid S' = \mathrm{Post}_\alpha(S_0)\nearrow\}$;
$Win[S_0] \leftarrow S_0 \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[S_0] \leftarrow \emptyset$;

Main:
while $((Waiting \neq \emptyset) \wedge (S_0 \notin Win[S_0]))$ do
  $e = (S, \alpha, S') \leftarrow pop(Waiting)$;
  if $S' \notin Passed$ then
    $Passed \leftarrow Passed \cup \{S'\}$;
    $Depend[S'] \leftarrow \{(S, \alpha, S')\}$;
    $Win[S'] \leftarrow S \cap (\{\mathrm{Goal}\} \times \mathbb{R}^X_{\geq 0})$;
    $Waiting \leftarrow Waiting \cup \{(S', \alpha, S'') \mid S'' = \mathrm{Succ}_\alpha(S')\}$;
    if $Win[S'] \neq \emptyset$ then $Waiting \leftarrow Waiting \cup \{e\}$;
    endif
  else (* reevaluate *)
    $Win^* \leftarrow \mathrm{Pred}_t(Win[S] \cup \bigcup_{S \xrightarrow{c} T} \mathrm{Pred}_c(Win[T]),$
                  $\bigcup_{S \xrightarrow{u} T} \mathrm{Pred}_u(T \setminus Win[T])) \cap S$;
    if $(Win[S] \subsetneq Win*)$ then
      $Waiting \leftarrow Waiting \cup Depend[S]$;
      $Win[S] \leftarrow Win^*$;
    endif
    $Depend[S'] \leftarrow Depend[S'] \cup \{e\}$;
  endif
endwhile
```

Fig. 1. Symbolic On-the-fly Algorithm for Timed Reachability Games, Cassez et al. (2005)

The algorithm requires some specific operations on symbolic states. We straightforwardly extend them for this extended notion of symbolic state.

Definition 11. (timed successors, timed predecessors, reset). Let Z be a set of valuations (w, v) for the clocks and the parameters. We define:

- timed successors $Z\nearrow = \{(w', v) \mid \exists t \geq 0 \wedge \exists (w, v) \in Z \text{ s.t. } w' = w + t\}$
- timed predecessors $Z\swarrow = \{(w', v) \mid \exists t \geq 0 \wedge \exists (w, v) \in Z \text{ s.t. } w' = w - t \wedge \forall x_i \in X,\ w(x_i) \leq 0\}$
- reset $Z[R] = \{(w', v) \mid \exists (w, v) \in Z \text{ s.t. } w' = w[R]\}$

Note that if the set of parameters is empty, we have exactly the usual definition of timed successors, timed predecessors and reset.

Let $[\![Z]\!]_v$ define a (non-parametric) zone obtained from a parametric zone Z for a fixed parameter valuation v.

Lemma 3. For any set of valuations on both clocks and parameters Z:

$$[\![Z\nearrow]\!]_v = [\![Z]\!]_v\nearrow \quad (1)$$
$$[\![Z\swarrow]\!]_v = [\![Z]\!]_v\swarrow \quad (2)$$
$$[\![Z[R]]\!]_v = [\![Z]\!]_v[R] \quad (3)$$

Here we state only the proof for timed successors, since the other two are done in the same way.

Proof. We will prove both inclusions:
1. $[\![Z\nearrow]\!]_v \subseteq [\![Z]\!]_v\nearrow$

Suppose that $w' \in [\![Z\nearrow]\!]_v$. Then $(w', v) \in Z\nearrow$ and by definition of a timed successor, $\exists t \geq 0$ such that $w' = w + t$ and $(w, v) \in Z$. Therefore $w \in [\![Z]\!]_v$, and then $\{w\}\nearrow \subseteq [\![Z]\!]_v\nearrow$. Since $w' = w + t \in \{w\}\nearrow$, we finally have $w' \in [\![Z]\!]_v$.

2. $[\![Z]\!]_v\nearrow \subseteq [\![Z\nearrow]\!]_v$

Suppose that $w' \in [\![Z]\!]_v \nearrow$. Then $\exists t \geq 0$ such that $w' = w + t$ and $w \in [\![Z]\!]_v$, i.e., $(w, v) \in Z$. By definition of a timed successor, this is exactly $(w', v) \in Z \nearrow$ or, equivalently, $w' \in [\![Z \nearrow]\!]_v$. □

For the union, intersection and difference set operations we only state the lemma:

Lemma 4. For any set of valuations on both clocks and parameters Z_1 and Z_2:

$$[\![Z_1 \cup Z_2]\!]_v = [\![Z_1]\!]_v \cup [\![Z_2]\!]_v \tag{4}$$

$$[\![Z_1 \cap Z_2]\!]_v = [\![Z_1]\!]_v \cap [\![Z_2]\!]_v \tag{5}$$

$$[\![Z_1 \backslash Z_2]\!]_v = [\![Z_1]\!]_v \backslash [\![Z_2]\!]_v \tag{6}$$

The zone successor by an edge, $Z' = Succ_\alpha(Z)$, in the simulation graph, is obtained, for a transition $(l, a, g, R, l') \in E$, by intersecting a source zone Z with the corresponding transition guard g, reseting clocks in reset set R, letting time elapse, and intersecting with target location invariants. The result, Z', is a zone as well, since zones are closed under intersection, reset, and timed successor operations.

Definition 12. (Zone successor). A zone successor by an edge, $Z' = Succ_\alpha(Z)$, is defined as:

$$Succ_\alpha(Z) = (Z \ \cap \ g)[R] \nearrow \cap I(l') \tag{7}$$

Using previous lemmas we can modify the zone successor operator for the parametric zones.

Lemma 5. For any set of valuations on both clocks and parameters Z:

$$[\![Succ_\alpha(Z)]\!]_v = Succ_\alpha([\![Z]\!]_v) \tag{8}$$

Proof. Following Lemma 3 and Lemma 4 : $[\![Succ_\alpha(Z)]\!]_v = [\![(Z \cap g)[R] \nearrow \cap I(l')]\!]_v = [\![(Z \cap g)[R] \nearrow]\!]_v \cap [\![I(l')]\!]_v = [\![(Z \cap g)[R]]\!]_v \nearrow \cap [\![I(l')]\!]_v = [\![(Z \cap g)]\!]_v[R] \nearrow \cap [\![Inv(l')]\!]_v = ([\![Z]\!]_v \cap [\![g]\!]_v)[R] \nearrow \cap [\![I(l')]\!]_v = Succ_\alpha([\![Z]\!]_v)$
□

Now we can modify, in the same manner, the operators needed for solving timed games.

Lemma 6. For any set on valuation on both clocks and parameters Z:

$$[\![Pred_a(Z)]\!]_v = Pred_a([\![Z]\!]_v) \tag{9}$$

Proof.

We will prove both inclusions:
1. $[\![Pred_a(Z)]\!]_v \subseteq Pred_a([\![Z]\!]_v)$

Suppose that $w \in [\![Pred_a(Z)]\!]_v$. Then $(w, v) \in Pred_a(Z)$ and by definition of action predecessor, $\exists w'$ such that $(l, w) \xrightarrow{a} (l', w')$ and $(w', v) \in Z$. Therefore $w' \in [\![Z]\!]_v$, and then $w \in Pred_a([\![Z]\!]_v)$.

2. $Pred_a([\![Z]\!]_v) \subseteq [\![Pred_a(Z)]\!]_v$

Suppose that $w' \in Pred_a([\![Z]\!]_v)$. Then $\exists w$ such that $(l', w') \xrightarrow{a} (l, w)$ and $w \in [\![Z]\!]_v$, i.e., $(w, v) \in Z$. By definition of action predecessor, we have $(w', v) \in Pred_a(Z)$, which gives us $w' \in [\![Pred_a(Z)]\!]_v$. □

The safe-timed predecessors operator can be expressed as (detailed proof in Cassez et al. (2005)):

$$Pred_t(Z_1, Z_2) = (Z_1 \swarrow \backslash Z_2 \swarrow) \cup ((Z_1 \cap Z_2 \swarrow) \backslash Z_2) \swarrow \tag{10}$$

Lemma 7. For any set of valuations on both clocks and parameters Z_1 and Z_2:

$$[\![Pred_t(Z_1, Z_2)]\!]_v = Pred_t([\![Z_1]\!]_v, [\![Z_2]\!]_v) \tag{11}$$

The proof is similar to the one of Lemma 5, and it applies to a safe-timed predecessor expressed by equation (10).

Now we have all the necessary operations modified for the parametric symbolic state-space, and we can prove the correctness of the parametric algorithm. Upon the termination of our semi-algorithm the following theorem stands:

Theorem 4. For a PGA $\mathcal{G}$, and a desired state goal, there exists a winning strategy for a parameter valuation function v if and only if $(l_0, w_0) \in [\![Win[S_0]]\!]_v$.

Proof. $Win(\mathcal{G})$ is a set of winning states in $\mathcal{G}$. The iterative process of the algorithm is given by $Win^0 = \mathsf{goal}$ and $Win^{n+1} = \pi(Win^n)$. We have that $Win^0 = [\![Win^0]\!]_v$, because $\mathsf{goal} = \{\mathsf{goal} \times \mathbb{R}^X_{\geq 0}\}$, and therefore is not affected by a parameter valuation function.

We apply a timed predecessors operator in a parametric domain and by Lemma 7 we have: $\pi([\![Win^n]\!]_v) = [\![\pi(Win^n)]\!]_v = [\![Win^{(n+1)}]\!]_v$. The least fix point obtained is $[\![Win^*]\!]_v$, and $[\![Win^*]\!]_v = [\![Win(\mathcal{G})]\!]_v$ (proved in Maler et al. (1995)).

A state (l, w) is winning if it belongs to $[\![Win(\mathcal{G})]\!]_v$, thus there is a winning strategy for $\mathcal{G}$ if $(l_0, w_0) \in [\![Win(\mathcal{G})]\!]_v$.

An invariance property of the algorithm for solving timed games Cassez et al. (2005), $Win[\![S]\!]_v \subseteq Win([\![\mathcal{G}]\!]_v)$, has been proven in Cassez et al. (2005). We can adapt the proof for the parametric case and show that $[\![Win[S]]\!]_v \subseteq [\![Win(\mathcal{G})]\!]_v$, for some $S = (l, Z)$, where Z is a parametric zone. We have that $Win[\![S]\!]_v = [\![Win[S]]\!]_v$, because $Win[S] \subseteq S$. By the induction hypothesis, we may assume, in the non-parametric case, that $Win[\![S']\!]_v \subseteq Win[\![\mathcal{G}]\!]_v$, when $S \xrightarrow{a} S'$. Zone Z', of the state S', is the successor of the zone Z. Since we have, by Lemma 5, that $Succ_\alpha([\![Z]\!]_v) = [\![Succ_\alpha(Z)]\!]_v$, we may also assume that $[\![Win[S']]\!]_v \subseteq [\![Win[\mathcal{G}]]\!]_v$. Then, by the monotonicity of $Pred_t$ it follows that $[\![Win^*]\!]_v \subseteq [\![(\pi(Win(\mathcal{G}))]\!]_v \subseteq [\![Win(\mathcal{G})]\!]_v$. Then, the property holds when we update $[\![Win^*]\!]_v \leftarrow [\![Win[S]]\!]_v$.

If $S = S_0$ and $(l_0, w_0) \in [\![Win[S_0]]\!]_v$ then $(l_0, w_0) \in [\![Win^*]\!]_v$, hence, there is a strategy to reach a goal state in $[\![\mathcal{G}]\!]_v$, and it can be extracted from $[\![Win^*]\!]_v$. □

The termination of our parametrization of the algorithm of Cassez et al. (2005) is not guaranteed. In the case of termination however, if the initial state belongs to a set of winning states, the correct set of constraints on the parameters is obtained and a winning strategy can be extracted from the set of winning states.

Example We consider the same example as in Cassez et al. (2005), but we parametrize the model in order to obtain a L/U game automaton (Figure 2). It has one clock x, controllable (c_i) and uncontrollable (u_i) actions and two parameters a and b: a appears positively in the guards of the controllable transitions c_1 and c_4 and negatively in the guard of the uncontrollable transition u_1; b appears positively in the guard of the uncontrollable transition u_3.

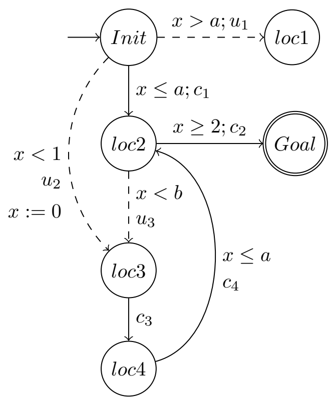


Fig. 2. A L/U Game Automaton

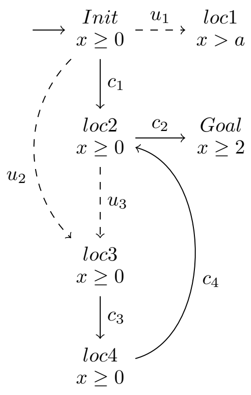


Fig. 3. Simulation graph of PGA of Figure2

The reachability game consists in finding a strategy for the controller that will eventually ends up in the location *Goal*.

We will now explain how the algorithm works. Although the algorithm of Cassez et al. (2005) is an interleaved combination of a forward computation and a backward propagation, for the sake of simplicity, we will start from a simulation-graph and show the back-propagation of winning states.

After the computation of a simulation graph, shown in Figure 3, the backward algorithm starts from the symbolic winning subset $(Goal, x \geq 2)$. By a controllable action (c_2) predecessor, we obtain $(loc2, x \geq 2)$. Computing the timed predecessors removes the constraint $x \geq 2$, and computing the *controllable predecessors* adds $x \geq b$ in order not to end-up in loc_3 by $u3$. The resulting state is $(loc2, x \geq b)$. One of the controllable transitions taking us to loc_2 is c_4. A controllable action predecessor (c_4) adds a constraint $x \leq a$. A constraint on the parameters derived in this state is $a \geq b$. This contraint is back-propagated to the preceding states. The (safe) timed predecessors give us the state $(loc4, x \geq 0 \wedge a \geq b)$.

We obtain successively the following sets of winning states: $(loc3, x \geq 0 \wedge a \geq b)$, $(loc2, (x \geq b) \vee (x \geq 0 \wedge a \geq b))$ and $(Init, (x \leq a) \wedge \big((x < 1 \wedge a \geq b) \vee x \geq 1\big) \wedge \big((x \geq b) \vee (x \geq 0 \wedge a \geq b)\big)$. The last one simplifies to $(Init, (x \leq a \wedge a \geq b))$.

4.3 Winning Strategy

In this section we show how to extract the winning strategy from the set of winning states.

Theorem 5. If there exists a winning strategy for the parametric timed game automaton $\mathcal{G}$, then there exists a memory-less winning strategy for $\mathcal{G}$.

Proof. If there exists a winning strategy for the parametric timed game automaton then there exists a parameter valuation v such that this winning strategy exists. For this parameter valuation v, we obtain a timed game automaton $[\![\mathcal{G}]\!]_v$, for which the winning strategy obtained by the algorithm is memory-less, since it suggests to a controller to either delay or take a controllable action in each state, Maler et al. (1995). □

Thanks to this theorem, it is easy to extract the memory-less winning strategy from the set of winning states as follows: a controllable action predecessor give us the state from which a corresponding controllable action should be taken, while timed predecessor further gives us the state where we should delay.

Let us go back to the previous example. The set of states $(loc2, x \geq 2)$ is the controllable action predecessor from $(Goal, x \geq 2)$ by action c_2. Then the winning strategy is: in all states $(loc2, x \geq 2)$ the controllable transition c_2 should be taken immediately, and in $(loc2, x \geq 0)$, we should delay until $x \geq 2$. The controllable action predecessor from $loc2$ takes us to a state $(loc4, x \geq b \wedge x \leq a)$, deriving a constraint $a \geq b$. From that state an action c_2 should be taken immediately, and timed predecessor gives the state $(loc4, x \geq 0, a \geq b)$ in which we should delay until $x \geq b$.

Since we can not influence the uncontrollable transitions if we end-up in $loc4$, controllable transition c_4 should be taken as soon as $x \geq b$ is satisfied, and with the condition $a \geq b$ we are sure that u_3 cannot be fired again.

Notice that the order of exploration of the winning states leads to different winning strategies. As an example, applying controllable predecessor from $(loc2, (x \geq b) \vee (x \geq 0 \wedge a \geq b))$ to *Init* can lead to both strategies from *Init*:

(1) doing c_1 in all states $(Init, x)$ with $x \leq a$;
(2) delaying in all states $(Init, x)$ with $x < b$ and $x \leq a$ and doing c_1 for all states with $x \geq b$ and $x \leq a$ (recall that $b \leq a$).

Thus, a whole winning strategy consists in:

- doing c_1 in all states $(Init, x)$ with $x \leq a$
- delaying in all states (loc_2, x) with $x < 2$
- doing c_2 in all states (loc_2, x) with $x \geq 2$
- doing c_3 in all states (loc_3, x)
- delaying in all states (loc_4, x) with $x < b$
- doing c_4 in all states (loc_4, x) with $x \geq b$ and $x \leq a$ (recall that $b \leq a$).

5. CASE STUDY

Let us consider the Copper Annealing Controller depicted in Figure 4. Annealing, in metallurgy and materials science, is a heat treatment wherein a material is altered, causing changes in its properties such as strength and hardness. It is a process that produces conditions by heating to above the critical temperature, maintaining a suitable temperature, and then cooling.

The parametric timed automaton shown in Figure 4 has two clocks x and y, two parameters a and b, controllable (c_i) and uncontrollable (u_i) actions. Action c_1 stops the heater to maintain the temperature. Actions c_2 and c_3 start and stop the cooler, respectively. The copper is observed by sensors that produce uncontrollable actions: u_1 is raised when the copper could be softer: it should be heated a bit more; u_2 is raised when the copper is too hard: the process must stop; u_3 is raised when the copper is soft enough: it should be cooled as soon as possible; u_4 is raised when the copper is too soft: the process must stop.

The parameter a means that a heating stage is followed by a maintaining stage whose duration is at least 10 time

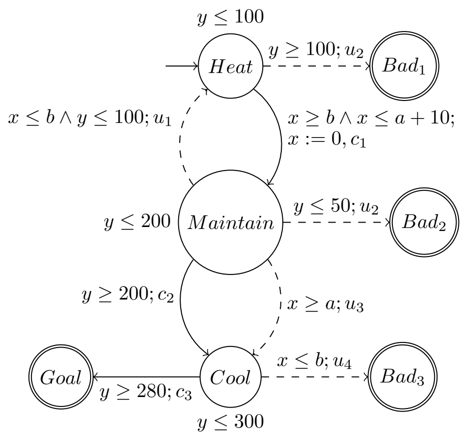


Fig. 4. A Copper Annealing Controller model

units longer than the heating duration. The parameter b comes from the dynamics of the system. For a copper wire heated during at least b time units, the values given by sensors u_1 and u_4 are relevant and guaranteed during b time units after the end of the heating stage.

The reachability game consists in finding a strategy, that will eventually end up in the location *Goal*. Actually, for this model, we obtain that there is no winning strategy for this game since it is impossible to prevent the transition u_1 from the locality $Maintain$ and then the locality Bad_1 is always reachable after some loops $(c_1.u_1)^*$ followed by u_2.

Then the model of the controller must be corrected. Since heating a bit more the copper, when it is possible, is not necessary, we can delete the transition u_1 (another way would consist in controlling the transition from $Maintain$ to $Heat$ when action u_1 occurs by adding a locality and two controllable actions). Thus, there exists a winning strategy if and only if $(b < 100) \wedge (a > 40) \wedge (a > b)$ and the set of winning states obtained by the algorithm is:

- $(Heat, (x \geq 0) \wedge (y \geq 0) \wedge (y < 100) \wedge (b < 100) \wedge (a > 40) \wedge (a > b))$,
- $(Maintain, (x \geq 0) \wedge (y > 50) \wedge (y \leq 200) \wedge (b \leq y - x \leq a + 10) \wedge (y - x < 100) \wedge (a > b))$,
- $(Cool, (x > b) \wedge (0 \leq y \leq 300) \wedge (b \leq y - x \leq a + 10) \wedge (y - x < 100))$,
- $(Goal, y \geq 280 \wedge (b \leq y - x \leq a + 10) \wedge (y - x < 100))$.

Intuitively, the condition $b < 100$ allows to avoid Bad_1 by ensuring that the locality *Heat* can be left before the condition $y \geq 100$ becomes true; the condition $a > 40$ allows to avoid Bad_2 by ensuring that the locality $Maintain$ can be reached with $y > 50$ and the condition $a > b$ allows to avoid Bad_3. A winning strategy extracted from the winning set consists in:

- delaying in all states $(Heat, x, y)$ with $y \leq 50$ or $x < b$
- doing c_1 in all states $(Heat, x, y)$ with $y > 50$ and $x \geq b$
- delaying in all states $(Maintain, x, y)$ with $y < 200$
- doing c_2 in all states $(Maintain, x, y)$ with $y = 200$
- delaying in all states $(Cool, x, y)$ with $y < 280$
- doing c_3 in all states $(Cool, x, y)$ with $280 \leq y$

6. CONCLUSIONS AND FUTURE WORK

In this paper we have introduced parametric timed game automata and their subclasses (L/U and restricted L/U game automata), for which the emptiness problem of parametric timed reachability game is decidable. We have also adapted the algorithm for solving timed games from Cassez et al. (2005) for the parametric case. When the initial state is winning, a set of constraints on the parameters is obtained together with a set of winning states.

Although the subclass of PGA proposed might seem overly restricted, in practice it is not rare that only one or two values are parametrized and we search for all their possible valuations, as shown in our case-study.

Our model can be put in use for deciding timed alternating simulation in parametric domain, which serves to model real-time controller synthesis problems. Simulation can also be accompanied by a logical characterization, Bozzelli et al. (2009): if TGA $\mathcal{A}$ is simulated by TGA $\mathcal{B}$ then whenever a formula holds in $\mathcal{A}$, it also holds in $\mathcal{B}$. In Bulychev et al. (2009), a notion of weak alternating simulation, that preserves controllability with respect to ATCTL, between two timed game automata has been introduced. Problem of checking given notion of simulation has been reduced to solving timed reachability game and an on-the-fly algorithm for solving this game is proposed. In the parametric case, this algorithm could give the constraints on the parameters such that the weak alternating simulation holds between two timed game automata (for example, an abstract model and its refinement).

REFERENCES

Alur, R. and Dill, D. (1994). A theory of timed automata. *Theoretical Computer Science B*, 126, 183–235.

Alur, R., Henzinger, T.A., and Vardi, M.Y. (1993). Parametric real-time reasoning. In *ACM Symposium on Theory of Computing*, 592–601.

Asarin, E., Maler, O., Pnueli, A., and Sifakis, J. (1998). Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*. Elsevier.

Bozzelli, L., Pinchinat, S., and Legay, A. (2009). On timed alternating simulation for concurrent timed games. In *FSTTCS 2009*, Leibniz Int. Proceedings in Informatics.

Bulychev, P., Chatain, T., David, A., and Larsen, K. (2009). Efficient on-the-fly algorithm for checking alternating timed simulation. In *FORMATS '09*, 73–87.

Cassez, F., David, A., Fleury, E., Larsen, K., and Lime, D. (2005). Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR'05*, volume 3653 of *LNCS*.

Hune, T., Romijn, J., Stoelinga, M., and Vaandrager, F. (2002). Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53, 183–220.

Maler, O., Pnueli, A., and Sifakis, J. (1995). On the synthesis of discrete controllers for timed systems. In *STACS '95*.

Ramadge, P.J.G. and Wonham, W.M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 81–98.

Robust Reachability in Timed Automata: A Game-based Approach [⋆]

Patricia Bouyer, Nicolas Markey, and Ocan Sankur

LSV, CNRS & ENS Cachan, France.
{bouyer,markey,sankur}@lsv.ens-cachan.fr

Abstract. Reachability checking is one of the most basic problems in verification. By solving this problem, one synthesizes a strategy that dictates the actions to be performed for ensuring that the target location is reached. In this work, we are interested in synthesizing "robust" strategies for ensuring reachability of a location in a timed automaton; with "robust", we mean that it must still ensure reachability even when the delays are perturbed by the environment. We model this perturbed semantics as a game between the controller and its environment, and solve the parameterized robust reachability problem: we show that the existence of an upper bound on the perturbations under which there is a strategy reaching a target location is EXPTIME-complete.

1 Introduction

Timed automata [2] are a timed extension of finite-state automata. They come with an automata-theoretic framework to design, model, verify and synthesize systems with timing constraints. One of the most basic problems in timed automata is the reachability problem: given a timed automaton and a target location, is there a path that leads to that location? This can be rephrased in the context of control as follows: is there a *strategy* that dictates how to choose time delays and edges to be taken so that a target location is reached? This problem has been solved long ago [2], and efficient algorithms have then been developed and implemented [13, 18].

However, the abstract model of timed automata is an idealization of real timed systems. For instance, we assume in timed automata that strategies can choose the delays with arbitrary precision. In particular, the delays can be arbitrarily close to zero (the system is arbitrarily fast), and clock constraints can enforce exact delays (time can be measured exactly). Although these assumptions are natural in abstract models, they need to be justified after the design phase. Indeed the situation is different in real-world systems: digital systems have response times that may not be negligible, and control software cannot ensure timing constraints exactly, but only up to some error, caused by clock imprecisions, measurement errors, and communication delays. A good control software must be *robust*, *i.e.*, it must ensure good behavior in spite of small imprecisions [11, 12].

[⋆] This work has been partly supported by project ImpRo (ANR-10-BLAN-0317)

In this work, we are interested in the synthesis of robust strategies in timed automata for reachability objectives, taking into account response times and imprecisions. We propose to model the problem as a game between a controller (that will guide the system) and its environment. In our semantics, which is parameterized by some $0 < \delta_P \leq \delta_R$, the controller chooses to delay an amount $d \geq \delta_R$, and the system delays d', where d' is chosen by the environment satisfying $|d - d'| \leq \delta_P$. We say that a given location is *robustly reachable* if there exist parameters $0 < \delta_P \leq \delta_R$ such that the controller has a winning strategy ensuring that the location is reached against any strategy of the environment. If δ_P and δ_R are fixed, this can be solved using techniques from control theory [3]. However δ_P, δ_R are better seen as parameters here, representing imprecisions in the implementation of the system (they may depend on the digital platform on which the system is implemented), and whose values may not be available in the design phase. To simplify the presentation, but w.l.o.g., we assume in this paper that $\delta = \delta_P = \delta_R$; our algorithm can easily be adapted to the general case (by adapting the shrink operator in Section 3).

Note that this semantics was studied in [6] for timed games with *fixed* parameters, where the parameterized version was presented as a challenging open problem. We solve this problem for reachability objectives in timed automata: we show that deciding the existence of $\delta > 0$, and of a strategy for the controller so as to ensure reachability of a given location (whatever the imprecision, up to δ), is $\mathsf{EXPTIME}$-complete. Moreover, if there is a strategy, we can compute a *uniform* one, which is parameterized by δ, using *shrunk difference bound matrices* (shrunk DBMs) that we introduced recently [17]. In this case, our algorithm provides a bound $\delta_0 > 0$ such that the strategy is correct for all $\delta \in [0, \delta_0]$. Our strategies also give quantitative information on how perturbations accumulate or can compensate. Technically, our work extends shrunk DBMs by *constraints*, and establishes non-trivial algebraic properties of this data structure (Section 3). The main result is then obtained by transforming the infinite-state game into a finite abstraction, which we prove can be used to symbolically compute a winning strategy, if any (see Section 4).

By lack of space, technical proofs have been omitted; they can be found in [5].

2 Robust reachability in timed automata

2.1 Timed automata and robust reachability

Given a finite set of clocks $\mathcal{C}$, we call *valuations* the elements of $\mathbb{R}_{\geq 0}^{\mathcal{C}}$. For a subset $R \subseteq \mathcal{C}$ and a valuation v, $v[R \leftarrow 0]$ is the valuation defined by $v[R \leftarrow 0](x) = v(x)$ for $x \in \mathcal{C} \setminus R$ and $v[R \leftarrow 0](x) = 0$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$ and a valuation v, the valuation $v + d$ is defined by $(v + d)(x) = v(x) + d$ for all $x \in \mathcal{C}$. We extend these operations to sets of valuations in the obvious way. We write $\mathbf{0}$ for the valuation that assigns 0 to every clock.

An atomic clock constraint is a formula of the form $k \preceq x \preceq' l$ or $k \preceq x - y \preceq' l$ where $x, y \in \mathcal{C}$, $k, l \in \mathbb{Z} \cup \{-\infty, \infty\}$ and $\preceq, \preceq' \in \{<, \leq\}$. A *guard* is a conjunction of atomic clock constraints. A valuation v satisfies a guard g, denoted $v \models g$, if all constraints are satisfied when each $x \in \mathcal{C}$ is replaced with $v(x)$.

Definition 1 ([2]). *A* timed automaton $\mathcal{A}$ *is a tuple* $(\mathcal{L}, \mathcal{C}, \ell_0, E)$*, consisting of finite sets* $\mathcal{L}$ *of locations,* $\mathcal{C}$ *of clocks,* $E \subseteq \mathcal{L} \times \Phi_{\mathcal{C}} \times 2^{\mathcal{C}} \times \mathcal{L}$ *of edges, and where* $\ell_0 \in \mathcal{L}$ *is the initial location. An edge* $e = (\ell, g, R, \ell')$ *is also written as* $\ell \xrightarrow{g,R} \ell'$*.*

Standard semantics of timed automata is usually given as a timed transition system. To capture robustness, we define the semantics as a game where perturbations in delays are uncontrollable. Given a timed automaton $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, E)$ and $\delta > 0$, we define the *perturbation game* of $\mathcal{A}$ w.r.t. δ as a two-player turn-based timed game $\mathcal{G}_\delta(\mathcal{A})$ between players *Controller* and *Perturbator*. The state space of $\mathcal{G}_\delta(\mathcal{A})$ is partitioned into $V_C \cup V_P$ where $V_C = \mathcal{L} \times \mathbb{R}_{\geq 0}^{\mathcal{C}}$ is the set of states that belong to Controller and $V_P = \mathcal{L} \times \mathbb{R}_{\geq 0}^{\mathcal{C}} \times \mathbb{R}_{\geq 0} \times E$ is the set of states that belong to Perturbator. The initial state is $(\ell_0, \mathbf{0})$ and belongs to Controller. The transitions are defined as follows: from any state $(\ell, v) \in V_C$, there is a transition to $(\ell, v, d, e) \in V_P$ whenever $d \geq \delta$, $e = (\ell, g, R, \ell')$ is an edge such that $v + d \models g$. Then, from any such state $(\ell, v, d, e) \in V_P$, there is a transition to $(\ell', (v + d + \epsilon)[R \leftarrow 0]) \in V_C$, for any $\epsilon \in [-\delta, \delta]$.

We assume familiarity with basic notions in game theory, and quickly survey the main definitions. A run in $\mathcal{G}_\delta(\mathcal{A})$ is a finite or infinite sequence of consecutive states starting at $(\ell_0, \mathbf{0})$. It is said maximal if it is infinite or cannot be extended. A strategy for Controller is a function that assigns to every non-maximal run ending in some $(\ell, v) \in V_C$, a pair (d, e) where $d \geq \delta$ and e is an edge enabled at $v + d$ (i.e., there is a transition from (ℓ, v) to (ℓ, v, d, e)). A run ρ is compatible with a strategy f if for every prefix ρ' of ρ ending in V_C, the next transition along ρ after ρ' is given by f. Given a target location ℓ, a strategy f is winning for the reachability objective defined by ℓ whenever all maximal runs that are compatible with f visit ℓ.

Observe that we require at any state (ℓ, v), that Controller should choose a delay $d \geq \delta$ and an edge e that is enabled after the chosen delay d. The edge chosen by Controller is always taken but there is no guarantee that the guard will be satisfied exactly when the transition takes place. In fact, Perturbator can perturb the delay d chosen by Controller by any amount $\epsilon \in [-\delta, \delta]$, including those that do not satisfy the guard. Notice that $\mathcal{G}_0(\mathcal{A})$ corresponds to the standard (non-robust) semantics of $\mathcal{A}$. We are interested in the following problem.

Problem 1 (Parameterized Robust Reachability). Given a timed automaton $\mathcal{A}$ and a target location ℓ, decide whether there exists $\delta > 0$ such that Controller has a winning strategy in $\mathcal{G}_\delta(\mathcal{A})$ for the reachability objective ℓ.

Notice that we are interested in the parameterized problem: δ is not fixed in advance. For fixed parameter, the problem can be formulated as a usual timed game, see [6]. Our main result is the decidability of this parameterized problem. Moreover, if there is a solution, we compute a strategy represented by parameterized difference-bound matrices where δ is the parameter; the strategy is thus *uniform* with respect to δ. In fact, we provide a bound $\delta_0 > 0$ such that the strategy is winning for Controller for *any* $\delta \in [0, \delta_0]$. These strategies also provide a quantitative information on how much the perturbation accumulates (See Fig. 3). The main result of this paper is the following:

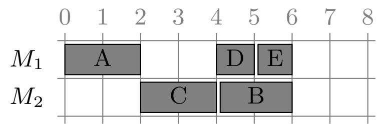
Theorem 2. *Parameterized robust reachability is* EXPTIME*-complete.*

Checking parameterized robust reachability is different from usual reachability checking mainly for two reasons. First, in order to reach a given location, Controller has to choose the delays along a run, so that these perturbations do not accumulate and block the run. In particular, it shouldn't play too close to the borders of the guards (see Fig. 3). Second, due to these uncontrollable perturbations, some regions that are not reachable in the absence of perturbation can become reachable (see Fig. 4). So, Controller must also be able to win from these new regions. The regions that become reachable in our semantics are those *neighboring* reachable regions. The characterization of these neighboring regions is one of the main difficulties in this paper (see Section 3.5).

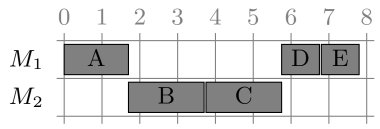
2.2 Motivating example: robust real-time scheduling

An application of timed automata is the synthesis of schedulers in various contexts [1]. We show that robust reachability can help providing a better schedulability analysis: we show that schedulers synthesized by standard reachability analysis may not be robust: even the slightest *decrease* in task execution times can result in a large *increase* in the total time. This is a phenomenon known as *timing anomalies*, first identified in [9].

Consider the scheduling problem described in Fig. 1, inspired by [16]. Assume that we look for a *greedy* (*i.e.*, work-conserving) scheduler, that will immediately start executing a task if a machine is free for execution on an available task. What execution time can guarantee a greedy scheduling policy on this instance? One can model this problem as a timed automaton, and prove, by classical reachability analysis, that these tasks can be scheduled using a greedy policy within six time units. However the scheduler obtained this way may not be robust, as illustrated in Fig. 1(b). If the duration of task A unexpectedly drops by a small amount $\delta > 0$, then any greedy scheduler will schedule task B before task C, since the latter is not ready for execution at time $2 - \delta$. This yields a scheduling of tasks in $8 - \delta$ time units.



(a) A has duration 2.



(b) A has duration $2 - \delta$.

Fig. 1. Consider tasks A, B, C of duration 2 and D, E of duration 1. Dependences between tasks are as follows: $A \to B$ and $C \to D, E$, meaning *e.g.* that A must be completed before B can start. Task A must be executed on machine M_1 and tasks B, C on machine M_2. Moreover, task C cannot be scheduled before 2 time units (which could be modelled using an extra task). Fig. 1(a) shows the optimal greedy schedule for these tasks under these constraints, while Fig. 1(b) shows the outcome of any greedy scheduler when the duration of task A is less than 2.

Our robust reachability algorithm is able to capture such phenomena, and can provide correct and robust schedulers. In fact, it would answer that the tasks are not schedulable in six time units (with a greedy policy), but only in eight time units.

2.3 Related work: robustness in timed automata and games

There has been a recent effort to consider imprecisions inherent to real systems in the theory of timed systems. In particular there has been several attempts to define convenient notions of robustness for timed automata, see [14] for a survey.

The approach initiated in [15, 8, 7] is the closest to our framework/proposition. It consists in *enlarging* all clocks constraints of the automaton by some parameter δ, that is transforming each constraint of the form $x \in [a, b]$ into $x \in [a-\delta, b+\delta]$, and in synthesizing $\delta > 0$ such that all runs of the enlarged automaton satisfy a given property. This can be reformulated as follows: does there exists some $\delta > 0$ such that whatever Controller and Perturbator do in $\mathcal{G}_\delta(\mathcal{A})$, a given property is satisfied. This is therefore the universal counterpart of our formulation of the parameterized robustness problem. It has been shown that this universal parameterized robust model-checking is no more difficult (in terms of complexity) than standard model-checking. This has to be compared with our result, where complexity goes up from PSPACE to EXPTIME.

Another work that is close to ours is that of [6]. The authors consider general two-player (concurrent) games with a fixed lower bound on delays, where chosen delays can be changed by some fixed value δ. It is then shown that winning strategies can be synthesized: In fact, when δ is fixed, the semantics can simply be encoded by a usual timed game, and standard algorithms can be applied. Whether one can synthesize $\delta > 0$ for which the controller has a winning strategy was left as a challenging open problem. We partially solve this open problem here, under the assumption that there is a single player with a reachability objective. The extension to two-player games (with reachability objective) is ongoing work, and we believe the techniques presented in this paper can be used for that purpose.

Finally, [10] studies a topological and language-based approach to robustness, where (roughly) a timed word is accepted by the automaton if, and only if, one of its neighborhoods is accepted. This is not related to our formalization.

3 Shrinking DBMs

3.1 Regions, zones and DBMs

We assume familiarity with the notions of regions and zones (see [4]). For two regions r and r', we write $r \lessdot r'$ if r' is the immediate (strict) time-successor of r. A *zone* is a set of clock valuations satisfying a guard.

We write $\mathcal{C}_0$ for the set $\mathcal{C} \cup \{0\}$. A *difference-bound matrix (DBM)* is a $|\mathcal{C}_0| \times |\mathcal{C}_0|$-matrix over $(\mathbb{R} \times \{<, \leq\}) \cup \{(\infty, <)\}$. A DBM M naturally represents a zone (which we abusively write M as well), defined as the set of valuations v

such that, for all $x, y \in \mathcal{C}_0$, writing $(M_{x,y}, \prec_{x,y})$ for the (x, y)-entry of M, it holds $v(x) - v(y) \prec_{x,y} M_{x,y}$ (where $v(0) = 0$). For any DBM M, let $\mathsf{G}(M)$ denote the graph over nodes $\mathcal{C}_0$, where the weight of the edge $(x, y) \in \mathcal{C}_0^2$ is $(M_{x,y}, \prec_{x,y})$. The normalization of M corresponds to assigning to each edge (x, y) the weight of the shortest path in $\mathsf{G}(M)$. We say that M is *normalized* when it is stable under normalization.

3.2 Shrinking

Consider the automaton $\mathcal{A}$ of Fig. 2, where the goal is to reach ℓ_3. If there is no perturbation or lower bound on the delays between transitions (*i.e.*, $\delta = 0$), then the states from which Controller can reach location ℓ_3 can be computed backwards. One can reach ℓ_3 from location ℓ_2 and any state in the zone $X = (x \leq 2) \wedge (y \leq 1) \wedge (1 \leq x - y)$, shown by (the union of the light and dark) gray areas on Fig. 3 (left); this is the set of time-predecessors of the corresponding guard. The set of winning states from location ℓ_1 is the zone $Y = (x \leq 2)$, shown in Fig. 3 (right), which is simply the set of predecessors of X at ℓ_2. When $\delta > 0$ however, the set of winning states at ℓ_2 is a "shrinking" of X, shown by the dark gray area. If the value of the clock x is too close to 2 upon arrival in ℓ_2, Controller will fail to satisfy the guard $x = 2$ due to the lower bound δ on the delays. Thus, the winning states from ℓ_2 are described by $X \cap (x \leq 2 - \delta)$. Then, this shrinking is backward propagated to ℓ_1: the winning states are $Y \cap (x \leq 2 - 2\delta)$, where we "shrink" Y by 2δ in order to compensate for a possible perturbation.

An important observation here is that when $\delta > 0$ is small enough, so that both $X \cap (x \leq 2 - \delta)$ and $Y \cap (x \leq 2 - 2\delta)$ are non-empty, these sets precisely describe the winning states. Thus, we have a *uniform* description of the winning states for "all small enough $\delta > 0$". We now define *shrunk DBMs*, a data structure we introduced in [17], in order to manipulate "shrinkings" of zones.

3.3 Shrunk DBMs

For any interval $[a, b]$, we define the *shrinking operator* as $\mathsf{shrink}_{[a,b]}(Z) = \{v \mid v + [a, b] \subseteq Z\}$ for any zone Z. We only use operators $\mathsf{shrink}_{[0,\delta]}$ and $\mathsf{shrink}_{[-\delta,\delta]}$ in the sequel. For a zone Z represented as a DBM, $\mathsf{shrink}_{[0,\delta]}(Z)$ is the DBM $Z - \delta \cdot \mathbb{1}_{\mathcal{C}\times\{0\}}$ and $\mathsf{shrink}_{[-\delta,\delta]}(Z)$ is the DBM $Z - \delta \cdot \mathbb{1}_{\mathcal{C}\times\{0\}\cup\{0\}\times\mathcal{C}}$, for any $\delta > 0$.

Our aim is to handle these DBMs symbolically. For this, we define *shrinking matrices (SM)*, which are nonnegative integer square matrices with zeroes on their diagonals. A *shrunk DBM* is then a pair (M, P) where M is a DBM, P is a

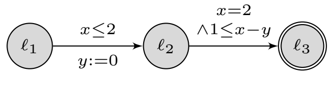


Fig. 2. Automaton $\mathcal{A}$

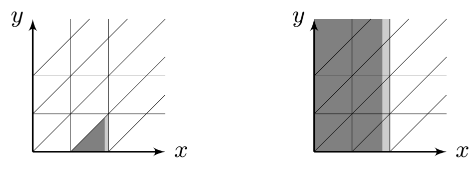


Fig. 3. Winning states in ℓ_2 (left) and in ℓ_1 (right)

shrinking matrix [17]. The meaning of this pair is that we consider DBMs $M - \delta P$ where $\delta \in [0, \delta_0]$ for some $\delta_0 > 0$. In the sequel, we abusively use "for all small enough $\delta > 0$" meaning "there exists $\delta_0 > 0$ such that for all $\delta \in [0, \delta_0]$". We also adopt the following notation: when we write a statement involving a shrunk DBM (M, P), we mean that the statement holds for $(M - \delta P)$ for all small enough $\delta > 0$. For instance, $(M, P) = \mathsf{Pre}_{\mathsf{time}}((N, Q))$ means that $M - \delta P = \mathsf{Pre}_{\mathsf{time}}((N - \delta Q))$ for all small enough $\delta > 0$. In the same vein, shrunk DBMs can be re-shrunk, and we write $\mathsf{shrink}((M, P))$ (resp. $\mathsf{shrink}^+((M, P))$) for the shrunk DBM (N, Q) such that $N - \delta Q = \mathsf{shrink}_{[-\delta,\delta]}(M - \delta P)$ (resp. $N - \delta Q = \mathsf{shrink}_{[0,\delta]}(M - \delta P)$) for all small enough $\delta > 0$.

It was shown in [17] that when usual operations are applied on shrunk DBMs, one always obtain shrunk DBMs, whose shrinking matrices can be computed. We refer to [4, 17] for the formal definitions of these operations.

Lemma 3 ([17]). *Let $M = f(N_1, \ldots, N_k)$ be an equation between normalized DBMs $M, N_1, \ldots, N_k$, using the operators $\mathsf{Pre}_{\mathsf{time}}$, $\mathsf{Unreset}_R$, $\cap$, shrink and shrink^+ and let $P_1, \ldots, P_k$ be SMs. Then, there exists a SM Q such that (M, Q) is normalized and $(M, Q) = f\big((N_1, P_1), \ldots, (N_k, P_k)\big)$. Moreover, Q and the corresponding upper bound on δ can be computed in polynomial time.*

3.4 Shrinking constraints

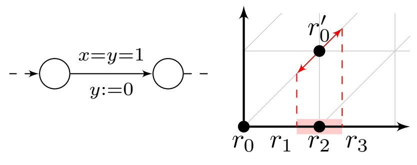


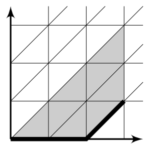
Fig. 4. Perturbing one transition

Consider a transition of a timed automaton, as depicted on the figure at right. From region r_0, the game can reach regions r_1, r_2, r_3, depending on the move of Perturbator. Therefore, in order to win, Controller needs a winning strategy from all three regions. One can then inductively look for winning strategies from these regions; this will generally require shrinking, as exemplified in Fig. 3. However, not all shrinkings of these regions provide a winning strategy from r_0. In fact, r_1 (resp. r_3) should not shrink from the right (resp. left) side: their union should include the shaded area, thus points that are arbitrarily close to r_2. In order to define the shrinkings that are useful to us, we introduce shrinking constraints.

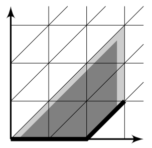
Definition 4. *Let M be a DBM. A* shrinking constraint *for M is a $|\mathcal{C}_0| \times |\mathcal{C}_0|$ matrix over $\{0, \infty\}$. A shrinking matrix P is said to* respect *a shrinking constraint S if $P \leq S$, where the comparison is component-wise. A pair $\langle M, S \rangle$ of a DBM and a shrinking constraint is called a* constrained DBM.

Shrinking constraints specify which facets of a given zone one is (not) allowed to shrink (see Fig. 5). A shrinking constraint S for a DBM M is said to be *well* if for any SM $P \leq S$, (M, P) is non-empty. A *well constrained DBM* is a constrained DBM given with a well shrinking constraint. We say that a shrinking constraint S for a DBM M is *normalized* if it is the minimum among all equivalent shrinking constraints: for any shrinking constraint S' if for all SMs

$$\begin{array}{c|ccc} & 0 & x & y \\ \hline 0 & (0,\le),0 & (0,<),\infty & (0,<),0 \\ x & (3,<),\infty & (0,\le),0 & (2,<),0 \\ y & (3,<),\infty & (0,<),\infty & (0,\le),0 \end{array}$$



(a) A constrained DBM $\langle M, S\rangle$ and its representation



(b) A shrinking of $\langle M, S\rangle$

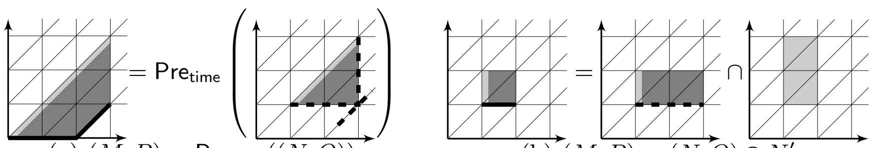
Fig. 5. Consider a zone defined by $0 < x < 3$, $0 < y < 3$, and $0 < x - y < 2$. Let the shrinking constraint S be defined by $S_{0,y} = 0$, $S_{x,y} = 0$, and $S_{z,z'} = \infty$ for other components. The resulting $\langle M, S\rangle$ is depicted on the left, as a matrix (where, for convenience, we merged both matrices into a single one) and as a constrained zone (where a thick segment is drawn for any boundary that is not "shrinkable", i.e., with $S_{z,z'} = 0$). On the right, the dark gray area represents a shrinking of M that satisfies S.

P, $P \le S \Leftrightarrow P \le S'$, then $S \le S'$. One can show that any shrinking constraint can be made normalized, by a procedure similar to the normalization of DBMs. Lemma 5 shows that shrinking constraints can be propagated along operations on DBMs. This is illustrated in Fig. 6.

Lemma 5. *Let M, N, N' be normalized non-empty DBMs.*

1. *Assume that $M = \mathsf{Pre}_{\mathsf{time}}(N)$, $M = N \cap N'$, or $M = \mathsf{Unreset}_R(N)$. Then, for any normalized well shrinking constraint S for M, there exists a well shrinking constraint S' for N such that for any SM Q, the following holds: $Q \le S'$ iff the SM P s.t. $(M, P) = \mathsf{Pre}_{\mathsf{time}}((N, Q))$ (respectively, $(M, P) = (N, Q) \cap N'$ or $(M, P) = \mathsf{Unreset}_R((N, Q))$) satisfies $P \le S$.*
2. *Assume that $M = N \cap N'$. For any well shrinking constraint S for N, there exists a shrinking constraint S' for M such that for any SM Q, the following holds: $Q \le S'$ iff a SM $P \le S$ s.t. $(N, P) \cap N' \subseteq (M, Q)$. Moreover, if $(N, P) \cap N' \ne \emptyset$ for all SMs $P \le S$, then S' is well.*

Let us comment on Fig. 6(a), and how it can be used for our purpose. Assume there is an edge guarded by N (the whole gray area in the right) without resets. In the non-robust setting, this guard can be reached from any point of M (the whole gray area in the left). If we have a shrinking constraint S on M, and we



(a) $(M, P) = \mathsf{Pre}_{\mathsf{time}}((N, Q))$ (b) $(M, P) = (N, Q) \cap N'$

Fig. 6. The figures illustrate the first item in Lemma 5. In each case, DBMs M, N and N' are fixed and satisfy the "unshrunk" equation. The thick plain segments represent the fixed shrinking constraint S. The dashed segments represent resulting constraint S'. For any SM Q, we have $Q \le S'$ iff there is an SM $P \le S$ that satisfies the equation.

want to synthesize a winning strategy from a shrinking of M satisfying S, then Lemma 5 gives the shrinking constraint S' for N, with the following property: given any shrinking (N, Q), we can find $P \leq S$ with $(M, P) = \mathsf{Pre}_{\mathsf{time}}((N, Q))$ (hence, we can delay into (N, Q)), if, and only if Q satisfies $Q \leq S'$. The problem is now "reduced" to finding a winning strategy from $\langle N, S' \rangle$. However, forward-propagating these shrinking constraints is not always that easy. We also need to deal with resets, with the fact that Controller has to choose a delay greater than $\delta > 0$, and also with the case where there are several edges leaving a location. This is the aim of the following developments.

3.5 Neighborhoods

We now consider constrained regions, which are constrained DBMs in which the DBM represents a region. Fig. 4 shows that if Controller plays to a region, then Perturbator can reach some of the surrounding regions, shown by the arrows. To characterize these, we define the set of *neighboring regions* of $\langle r, S \rangle$ as,

$$\mathcal{N}_{r,S} = \left\{ r' \;\middle|\; r' \lessdot^* r \text{ or } r \lessdot^+ r', \text{ and } \forall Q \leq S.\ r' \cap \mathsf{enlarge}((r, Q)) \neq \emptyset \right\}$$

where $\mathsf{enlarge}((r, Q))$ is the shrunk DBM (M, P) such that $v + [-\delta, \delta] \subseteq M - \delta P$ for every $v \in r - \delta Q$. This is the set of regions that have "distance" at most δ to any shrinking of the constrained region (r, S). We write $\mathsf{neighbor}\langle r, S \rangle = \bigcup_{r' \in \mathcal{N}_{r,S}} r'$.

Lemma 6 (Neighborhood). *Let $\langle r, S \rangle$ be a well constrained region. Then $\mathsf{neighbor}\langle r, S \rangle$ is a zone. If N is the corresponding normalized DBM, there exists a well shrinking constraint S' such that for every SM Q, $Q \leq S'$ iff the SM P defined by $(r, P) = r \cap \mathsf{shrink}((N, Q))$, satisfies $P \leq S$. The pair $\langle N, S' \rangle$ is the* constrained neighborhood *of $\langle r, S \rangle$, and it can be computed in polynomial time.*

Constrained neighborhoods are illustrated in Fig. 7.

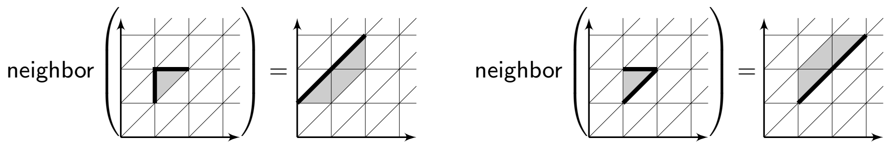


Fig. 7. Constrained neighborhood of two constrained regions. Notice that inside any shrinking of the constrained region, there is always a valuation such that a perturbation of $[-\delta, \delta]$ moves the valuation to any region of the neighborhood.

3.6 Two crucial properties for the construction of the abstraction

The following lemma characterizes, given a constrained region $\langle r, S \rangle$, the set of constrained regions $\langle r', S_{r'} \rangle$ such that any shrunk region satisying $\langle r', S_{r'} \rangle$ can be reached by delaying from some shrunk region satisfying $\langle r, S \rangle$.

Lemma 7. *Let $\langle r, S\rangle$ be a well constrained region, and r' be a region such that $r \lessdot^* r'$. Then the following properties are equivalent:*

1. *there exists a well shrinking constraint S' (which can be computed in polynomial time) such that for every SM Q, $Q \leq S'$ iff the SM P such that $(r, P) = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}((r', Q)))$, satisfies $P \leq S$;*
2. *$\mathsf{neighbor}\langle r, S\rangle \subseteq \mathsf{Pre}_{\mathsf{time}}(r')$;*

Note that this lemma may not hold for all r' with $r \lessdot r'$. Consider the constrained region $\langle r, S\rangle$ on the right of Fig. 7, and let r' be the first triangle region above r: any valuation arbitrarily close to the thick segments will be in $r - \delta P$ for any $P \leq S$, but it can only reach r' by delaying less than δ time units.

Lemma 8. *Let $\langle r, S\rangle$ be a well constrained region, and let $R \subseteq \mathcal{C}$. Let $\mathcal{N}$ be the set of neighboring regions of $\langle r, S\rangle$, and $\mathcal{N}' = \{r'[R \leftarrow 0] \mid r' \in \mathcal{N}\}$. Then, there exist well shrinking constraints $S_{r''}$ for all $r'' \in \mathcal{N}'$ such that for any $(Q_{r''})_{r'' \in \mathcal{N}'}$, we have $Q_{r''} \leq S_{r''}$ for all $r'' \in \mathcal{N}'$ iff there exists $P \leq S$ such that*

$$(r, P) \subseteq r \cap \mathsf{shrink}\Big(\bigcup_{r' \in \mathcal{N}} (r' \cap \mathsf{Unreset}_R((r'', Q_{r''})))\Big).$$

with $r'' = r'[R \leftarrow 0]$. Moreover, all $\langle r'', S_{r''}\rangle$ can be computed in polynomial time.

This lemma gives for instance the shrinking constraints that should be satisfied in r_1, r_2 and r_3, in Fig. 4, once shrinking constraint in r'_0 is known. In this case, the constraint in r'_0 is 0 everywhere since it is a punctual region. The neighborhood $\mathcal{N}$ of r'_0 is composed of r'_0 and two extra regions (defined by $(0 < x < 1) \wedge (x = y)$ and $(1 < x < 2) \wedge (x = y)$). If there are shrinkings of regions r_1, r_2, r_3 satisfying the corresponding shrinking constraints (given in the lemma), and from which Controller wins, then one can derive a shrinking of r'_0, satisfying its constraint, and from which Controller wins. In the next section, we define the game $\mathcal{RG}(\mathcal{A})$ following this idea, and explain how it captures the game semantics for robustness.

4 A finite game abstraction

Let $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, E)$ be a timed automaton. We define a finite turn-based game $\mathcal{RG}(\mathcal{A})$ on a graph whose nodes are of two sorts: *square nodes* labelled by (ℓ, r, S_r), where ℓ is a location, r a region, S_r is a well shrinking constraint for r; *diamond nodes* labelled similarly by (ℓ, r, S_r, e) where moreover e is an edge leaving ℓ. Square nodes belong to Controller, while diamond nodes belong to Perturbator. Transitions are defined as follows:

(a) From each square node (ℓ, r, S_r), for any edge $e = (\ell, g, R, \ell')$ of $\mathcal{A}$, there is a transition to the diamond node $(\ell, r', S_{r'}, e)$ if the following conditions hold:
  (i) $r \lessdot^* r'$ and $r' \subseteq g$;
  (ii) $S_{r'}$ is such that for all SMs Q, $Q \leq S_{r'}$ iff there exists $P \leq S_r$ with

$$(r, P) = r \cap \mathsf{shrink}^+(\mathsf{Pre}_{\mathsf{time}}((r', Q)))$$

(b) From each diamond node (ℓ, r, S_r, e), where $e = (\ell, g, R, \ell')$ is an edge of $\mathcal{A}$, writing $\mathcal{N}$ for the set of regions in the neighborhood of (r, S_r) and $\mathcal{N}' = \{r'[R \leftarrow 0] \mid r' \in \mathcal{N}\}$, there are transitions to all square nodes $(\ell', r'', S_{r''})$ with $r'' \in \mathcal{N}'$, and $(S_{r''})_{r'' \in \mathcal{N}'}$ are such that for all SMs $(Q_{r''})_{r'' \in \mathcal{N}'}$, it holds $Q_{r''} \leq S_{r''}$ for every $r'' \in \mathcal{N}'$ iff there exists $P \leq S_r$ such that

$$(r, P) \subseteq r \cap \mathsf{shrink}\Big(\bigcup_{r' \in \mathcal{N}} (r' \cap \mathsf{Unreset}_R((r'', Q_{r''}))\Big) \qquad (\text{where } r'' = r'[R \leftarrow 0])$$

Intuitively, the transitions from the square nodes are the decisions of Controller. In fact, it has to select a delay and a transition whose guard is satisfied. Then Perturbator can choose any region in the neighborhood of the current region, and, after reset, this determines the next state.

Note that $\mathcal{RG}(\mathcal{A})$ can be computed, thanks to Lemmas 7 and 8, and has exponential-size. Observe also that $\mathcal{RG}(\mathcal{A})$ is constructed in a forward manner: we start by the initial constrained region (*i.e.* the region of valuation $\mathbf{0}$ with the zero matrix as shrinking constraint), and compute its successors in $\mathcal{RG}(\mathcal{A})$. Then, if Controller has a winning strategy in $\mathcal{RG}(\mathcal{A})$, we construct a winning strategy for $\mathcal{G}_\delta(\mathcal{A})$ by a backward traversal of $\mathcal{RG}(\mathcal{A})$, using Lemmas 7 and 8. Thus, we construct $\mathcal{RG}(\mathcal{A})$ by propagating shrinking constraints forward, but later do a backward traversal in it. The correctness of the construction is stated as follows.

Proposition 9. *Controller has a winning strategy in $\mathcal{RG}(\mathcal{A})$ if, and only if there exists $\delta_0 > 0$ such that Controller wins $\mathcal{G}_\delta(\mathcal{A})$ for all $\delta \in [0, \delta_0]$.*

Note that as we compute a winning strategy for Controller (if any) by Proposition 9, we can also compute a corresponding δ_0. One can show, by a rough estimation, that $1/\delta_0$ is at worst doubly exponential in the size of $\mathcal{A}$.

Let us point out an interesting intermediary result of the proof: given a winning strategy for Perturbator in $\mathcal{RG}(\mathcal{A})$, we show that there is a winning strategy for Perturbator in $\mathcal{G}_\delta(\mathcal{A})$ that keeps the compatible runs close to borders of regions where shrinking constraints are 0.

The upper bound of Theorem 2 is a consequence of the above proposition, since $\mathcal{RG}(\mathcal{A})$ has exponential size and finite reachability games can be solved in time polynomial in the size of the game. The $\mathsf{EXPTIME}$ lower bound is obtained by simulating an alternating-time linear-bounded Turing machine. Simulation of the transitions is rather standard in timed-automata literature (though we must be careful here as delays can be perturbed). The difficult point is to simulate conjunctions: this is achieved using the module of Fig. 8. From the initial state, Controller has no choice but to play the first transition when $y = 1$. Perturbator can either anticipate or delay this transition, which will determine which of the dashed or dotted transitions is available next. This way, Perturbator decides by which end the module is exited.

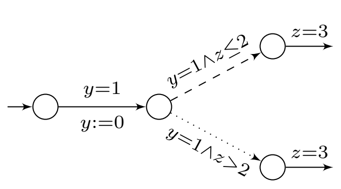


Fig. 8. Conjunction

5 Conclusion

We considered a game-based approach to robust reachability in timed automata. We proved that robust schedulers for reachability objectives can be synthesized, and that the existence of such a scheduler is EXPTIME-complete (hence harder than classical reachability [2]). We are currently working on a zone-based version of the algorithm, and on extending the techniques of this paper to the synthesis of robust controllers in timed games, which will answer an open problem posed in [6] for reachability objectives. Natural further works also include the synthesis of robust schedulers for safety objectives. This seems really challenging, and the abstraction we have built here is not correct in this case (it requires at least a notion of *profitable cycles à la* [7]). Another interesting direction for future work is to assume imprecisions are probabilistic, that is, once Controller has chosen a delay d, the real delay is chosen in a stochastic way in the interval $[d - \delta, d + \delta]$.

References

1. Y. Adbeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *TCS*, 354(2):272–300, 2006.
2. R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.
3. E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *SSSC'98*, pp. 469–474. Elsevier, 1998.
4. J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *ACPN'03*, vol. 2098 of *LNCS*, pp. 87–124. Springer, 2004.
5. P. Bouyer, N. Markey, and O. Sankur. Robust reachability in timed automata: A game-based approach. Technical Report LSV-12-07, Lab. Specification & Verification, ENS Cachan, France, May 2012.
6. K. Chatterjee, T. A. Henzinger, and V. S. Prabhu. Timed parity games: Complexity and robustness. *LMCS*, 7(4), 2010.
7. M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robust safety of timed automata. *FMSD*, 33(1-3):45–84, 2008.
8. M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. *FAC*, 17(3):319–341, 2005.
9. R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Applied Maths*, 17(2):416–429, 1969.
10. V. Gupta, T. A. Henzinger, and R. Jagadeesan. Robust timed automata. In *HART'97*, vol. 1201 of *LNCS*, pp. 331–345. Springer, 1997.
11. T. A. Henzinger and J. Sifakis. The embedded systems design challenge. In *FM'06*, vol. 4085 of *LNCS*, pp. 1–15. Springer, 2006.
12. H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Springer, 2011.
13. K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Intl J. STTT*, 1(1-2):134–152, 1997.
14. N. Markey. Robustness in real-time systems. In *SIES'11*, pp. 28–34. IEEE Comp. Soc. Press, 2011.
15. A. Puri. Dynamical properties of timed automata. *DEDS*, 10(1-2):87–113, 2000.
16. J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker. A definition and classification of timing anomalies. In *WCET'06*, 2006.
17. O. Sankur, P. Bouyer, and N. Markey. Shrinking timed automata. In *FSTTCS'11*, vol. 13 of *LIPIcs*, pp. 375–386. LZI, 2011.
18. S. Yovine. Kronos: A verification tool for real-time systems. *Intl J. STTT*, 1(1-2):123–133, 1997.